I/O BOOKS

「Writer」「Calc」「Base」…OOoをBasicで便利に！

OpenOffice マクロプログラミング

宍戸　輝光

Terumitsu Shishido

工学社

主な内容

■ マクロの基本
マクロとは
簡単なマクロの作成
ユーザー側からの入力内容を処理するマクロ
WriterやCalcの画面からマクロを実行する

■ マクロ・プログラミングの基礎
変数
変数を使った計算
数値と文字列を組み合わせた演算
関数
条件分岐
ループ処理①
数当てゲーム
配列
ループ処理②
多次元配列

■ OpenOfficeマクロに用意されている、便利な機能
CSVの処理
ファイルの読み書き
オブジェクト
インターフェイス
ダイアログの作成
イベント・ハンドラ
XMLの処理

■ マクロとアプリケーションの連携
Calcの表計算シートの扱い方
セルの扱い方
CalcのシートにGUI部品を配置
Calcのシートに図形を配置
Excelの「VBAマクロ」を使う
データベース「Base」

■ CSVファイルの「整形」「自動処理」
CSVファイルを特定の条件で処理する

■ 付録
「OpenOffice.org」のインストール
「OpenOffice.org」の派生ソフト「Go-OO」
困ったときのために

OpenOffice マクロプログラミング

はじめに

フリーのオフィス・ソフト「OpenOffice.org」には、「Basic」ベースのマクロ(OpenOffice.org Basic)環境が用意されており、気軽にプログラミングを楽しむことができます。

「Basic」というと"古い"感じもありますが、プログラミングの世界にはじめて入ろうとする人にとっては、むしろ分かりやすい言語と言えるかもしれません。

すっかり主流の地位を占めているC言語系の構文に比べて、プログラミングの世界特有の特別な記号や書き方が少ないので、最初の一歩で戸惑いにくいからです。

「JavaScriptを少しやってみたけれど、なじめなかった」という人も、一度、「OpenOffice.org Basic」を試してみてはいかがでしょうか。

「OpenOffice.org Basic」からは、表計算シートや文字列処理など「OpenOffice.org」の高度な機能を扱えるので、実用的なアプリ開発はもちろん、気軽にプログラミングを楽しむ入門環境としても魅力的です。

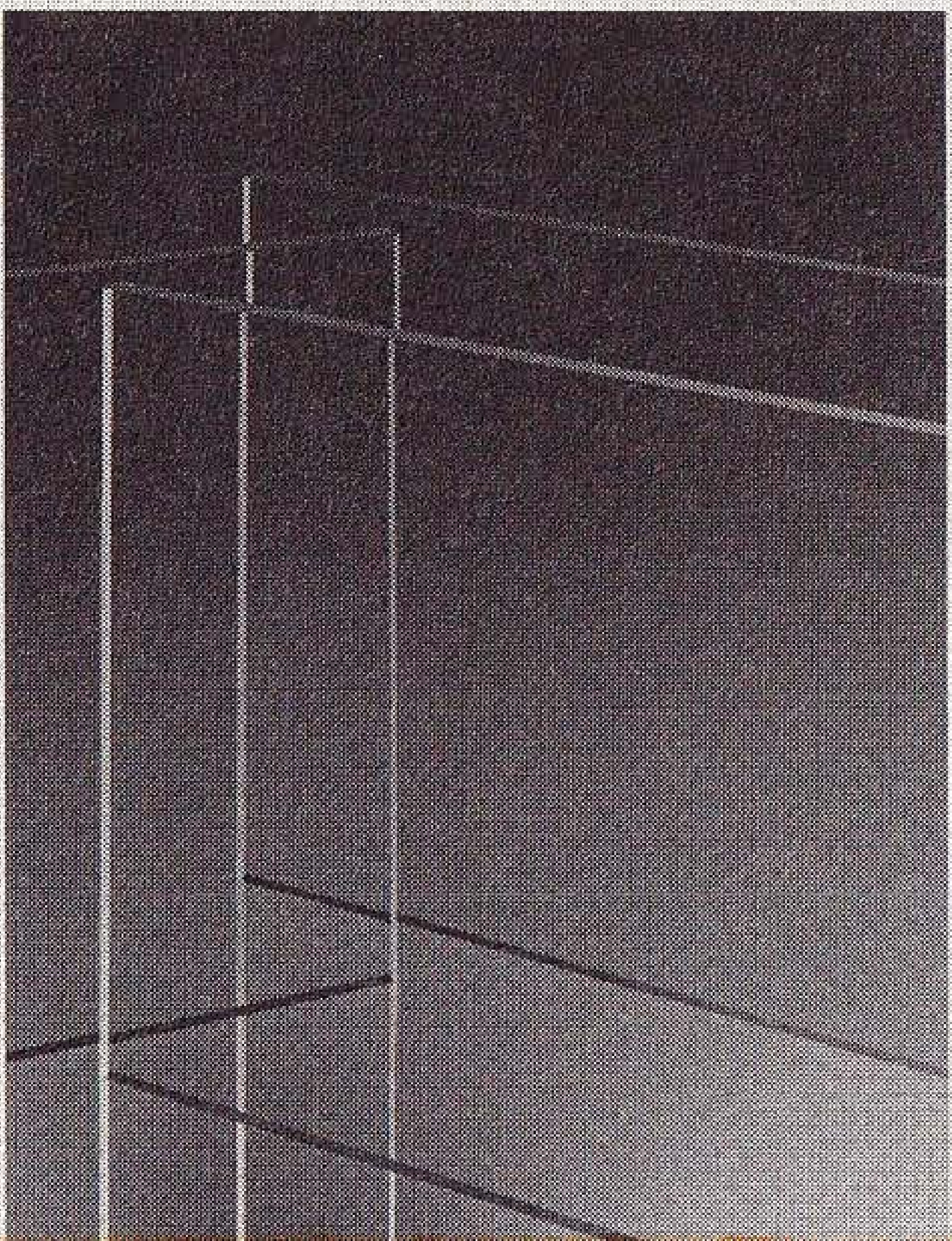
本書では、「Basic」の書き方や、「Calc」を中心とする「OpenOffice.org」の機能を、「Basic」から活用するための一連の流れをまとめてみました。

「OpenOffice.org」をさらに活用するために、また、ちょっとしたマクロを組みながら「自分でプログラムを書いて動かす」楽しみを味わうために、マクロに挑戦してみてください。

宍戸輝光

OpenOffice マクロプログラミング

CONTENTS



第3章 OpenOfficeマクロに用意されている、便利な機能

第4章 マクロとアプリケーションの連携

第5章 CSVファイルの「整形」「自動処理」

本書サンプル・プログラムについて

本書で解説しているサンプル・プログラムは、工学社のサポートページからダウンロードできます。

≪工学社ホームページ≫

http://www.kohgakusha.co.jp/

サポートページから、「sample.zip」をダウンロードしてください。

サンプル・プログラムはWindows版のOpenOffice.org3.1で動作を確認しています。

Mac版やLinux版、その他のバージョンでは、正しく動作しない可能性があります。あらかじめご了承ください。

【免責事項】

本書で取り扱っていないOpenOfficeマクロのプログラミング全般に関する質問の回答は、編集部では一切行なっておりません。それらに関しては、「OpenOffice.org日本語プロジェクト」の公式ページを利用してください。

【OpenOffice.org日本語プロジェクト】

http://ja.openoffice.org/

また、本書サンプルプログラムによって生じたいかなる障害についても、責任を負いかねます。使用に際しては、必ず自己責任で行なってください。

第1章

マクロの基本

この章では、「マクロとはどういったものなのか」や「OOoでのマクロの実行方法」など、「マクロ・プログラミング」の基本を解説します。
　実際に簡単なマクロを作ってみて、感覚を掴んでみましょう。

1-1 マクロとは

「OpenOffice.org」(OOo)には、**「マクロ」**と呼ばれるプログラミング機能があります。

マクロ機能は、「Microsoft Office」や各種グラフィック・ソフトなど、多くのソフトに用意されているので、他のソフトでマクロを使った経験をもっている人も多いかもしれません。

マクロを使うと、面倒な操作を一気に実行したり、独自の機能を追加できるので、ソフトの活用範囲を広げることができます。

「OOo」でもマクロ機能を使えば、各種の操作を自動的に行なう仕組みや、独自の新しい機能を追加できます。

「Calcのセルに自動的に値を入力する」「指定の条件で自動的にセルの背景を変える」「データ・ファイルからデータを読み込み自動的に処理する」といった「処理の自動化マクロ」を作ったり、複数の関数と演算を組み合わせて独自のマクロ関数を定義することで、"自分だけのより便利な「OOo」"に育てていくことができるわけです。

さらに、マクロからは「ボタン」「メニュー」「フォーム」など、各種の「GUIコンポーネント」を扱うこともできるので、「OOo」の機能を利用しながら本格的な「アプリケーション」を開発することもできます。

たとえば、

> データベースにアクセスし、GUIコンポーネントを駆使して作った入力フォームで情報の検索や編集を行なう。入力フォーム上に入力されたデータはマクロで検証を行ない、不適切な部分があれば、エラーを表示したり自動修正する

といった「データ管理アプリケーション」は、通常のプログラミング環境で開発しようと思うと、かなり手数がかかりますが、「OOo」のマクロを使えば、比較的簡単に開発できる場合があります。

「OOo」には、手軽に利用できる「データベース・アクセス機能」や、GUIコンポーネントにデータベースの内容を自動的に反映させる仕組みなどがあらかじめ用意されていて、それらの機能をマクロから簡単に利用できるからです。

これらのマクロは、「OOo」に用意されているプログラム開発環境(IDE)で、プログラムを作ることで追加できます。

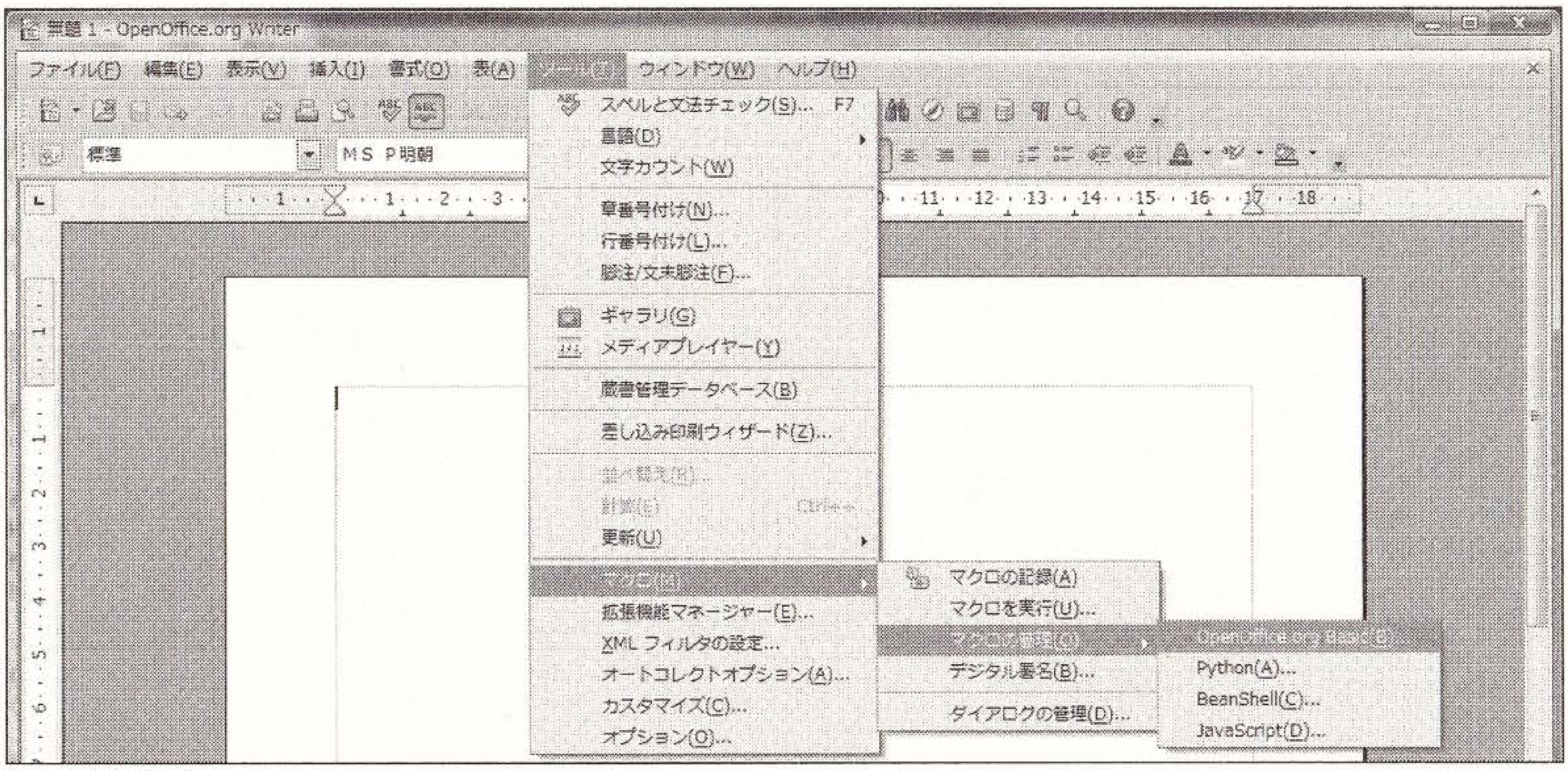
1-2 簡単なマクロの作成

では、実際に簡単なマクロを作ってみましょう。

■ マクロの編集画面の開き方

まず、マクロを実行するアプリケーションとして適当な「OOoアプリケーション」(「Calc」や「Writer」)を起動します。

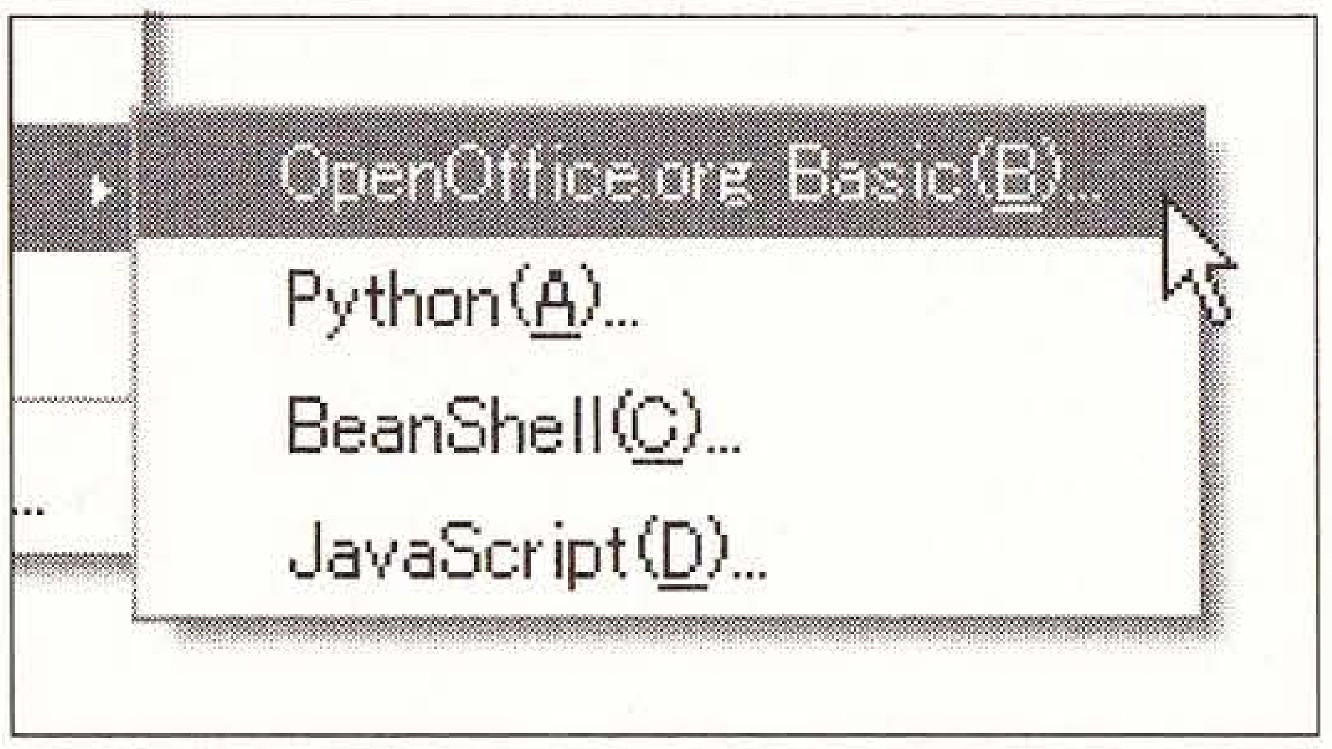
続いて、「ツール」メニューの「マクロ」→「マクロの管理」にマウス・カーソルを移動してみましょう。すると、「OpenOffice.org BASIC」「Python」「BeanShell」「JavaScript」といった項目が出てくると思います。

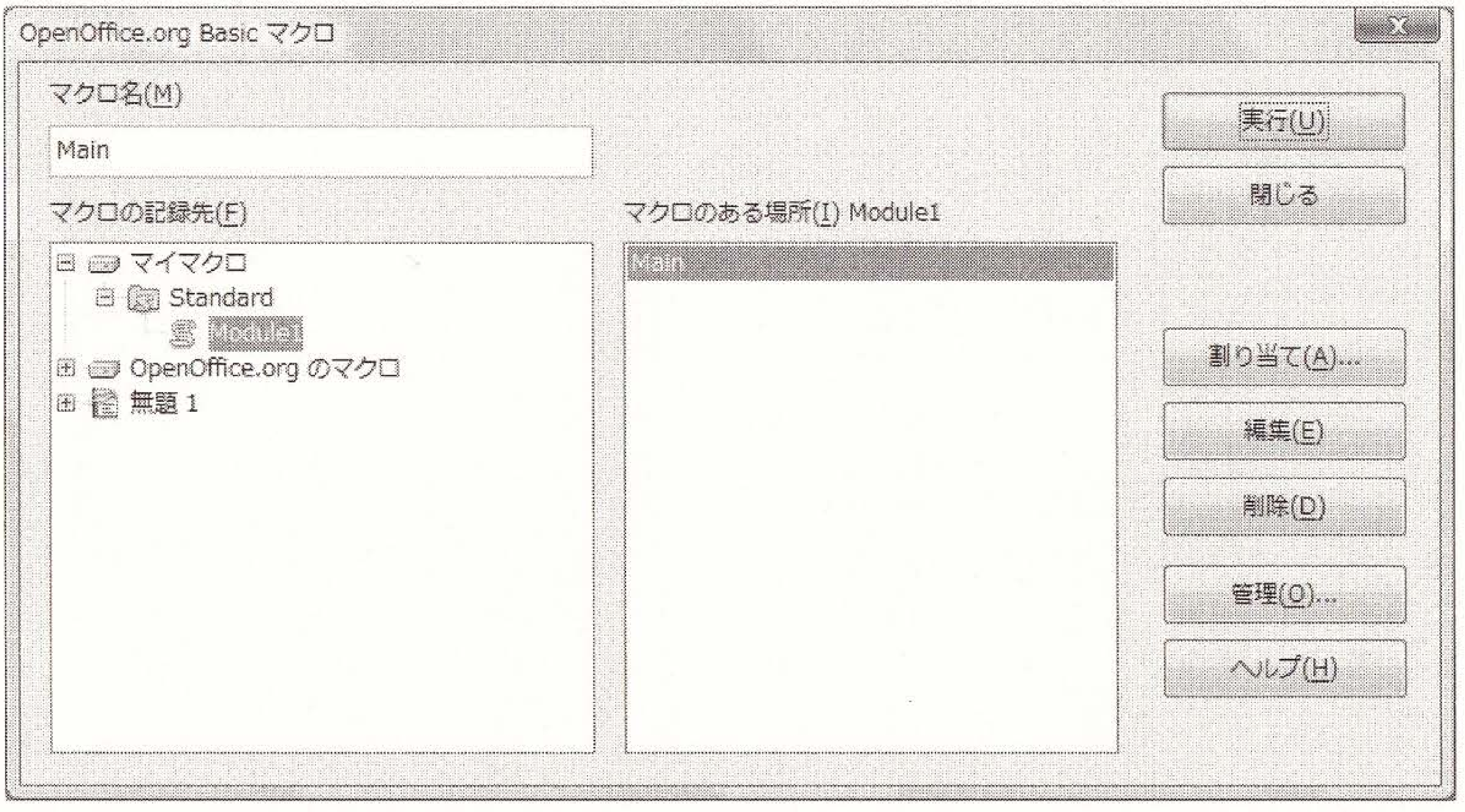


「マクロ」→「マクロの管理」を選択

これらは、「OOo」のマクロで利用できるプログラミング言語です。
本書では、比較的扱いやすい「OpenOffice.org BASIC」でマクロを作ることにします。

「OpenOffice.org BASIC」を選ぶと、マクロの管理画面になります。ここで編集するマクロを選んで編集画面に入るわけですが、今回はとりあえず文書ファイルにマクロを作ってみましょう。

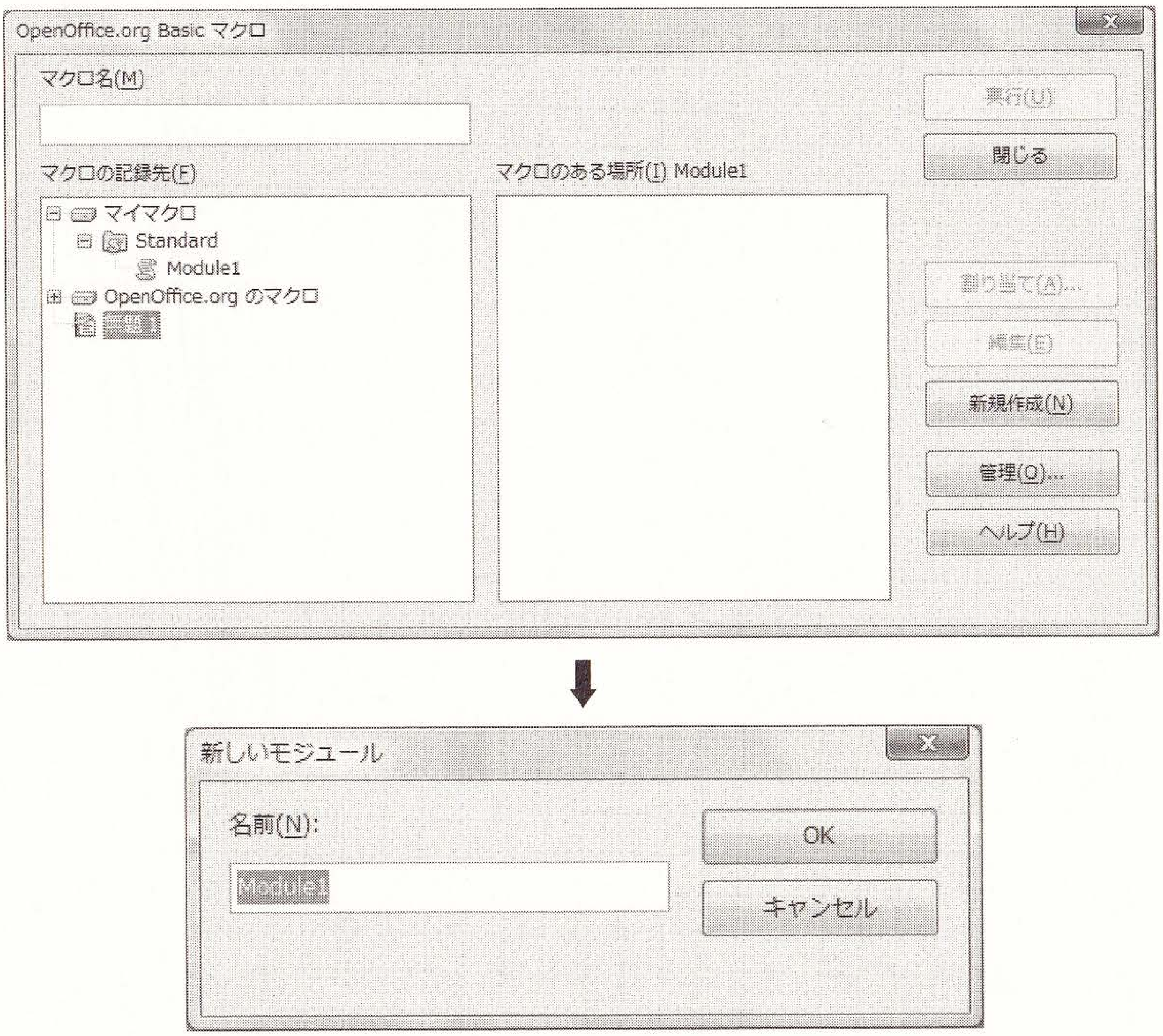




「OpenOffice.org BASIC」を選択し、マクロの管理画面を開く

「無題1」など、現在の文書の名前が付いたアイコンをクリックし、選択状態にしたら、「新規作成」ボタンをクリックしてください。

すると、「新しいモジュール」というモジュール名を指定するダイアログが出てきますが、ここはそのままOKをクリックします。



文書ファイル名のアイコン(画像では「無題1」)を選択した状態で、「新規作成」をクリック。その後に出てくる「新しいモジュール」は、そのまま「OK」をクリックする。

これで、実際にマクロを記述する編集画面になったと思います。

■ サブルーチン

編集画面には、

```
Sub Main

End Sub
```

というテキストがあります。これは、「Mainという名のサブルーチン」を定義するプログラムです。

この「Sub Main」と「End Sub」の間に行ないたい処理を書けば、その処理を「Main」という名の**「サブルーチン」**として登録できます。

「サブルーチン」とは、「ひとまとまりの処理」のことで、マクロを作る場合は必要に応じていくつかの「サブルーチン」(や後に出てくる「関数」)としてマクロを作り、そのサブルーチンを呼び出す形でマクロを実行します。



「Sub Main」と「End Sub」の間に目的の処理を書き込んでいく

■ メッセージの表示

手始めに何かメッセージを表示するマクロを作ってみましょう。

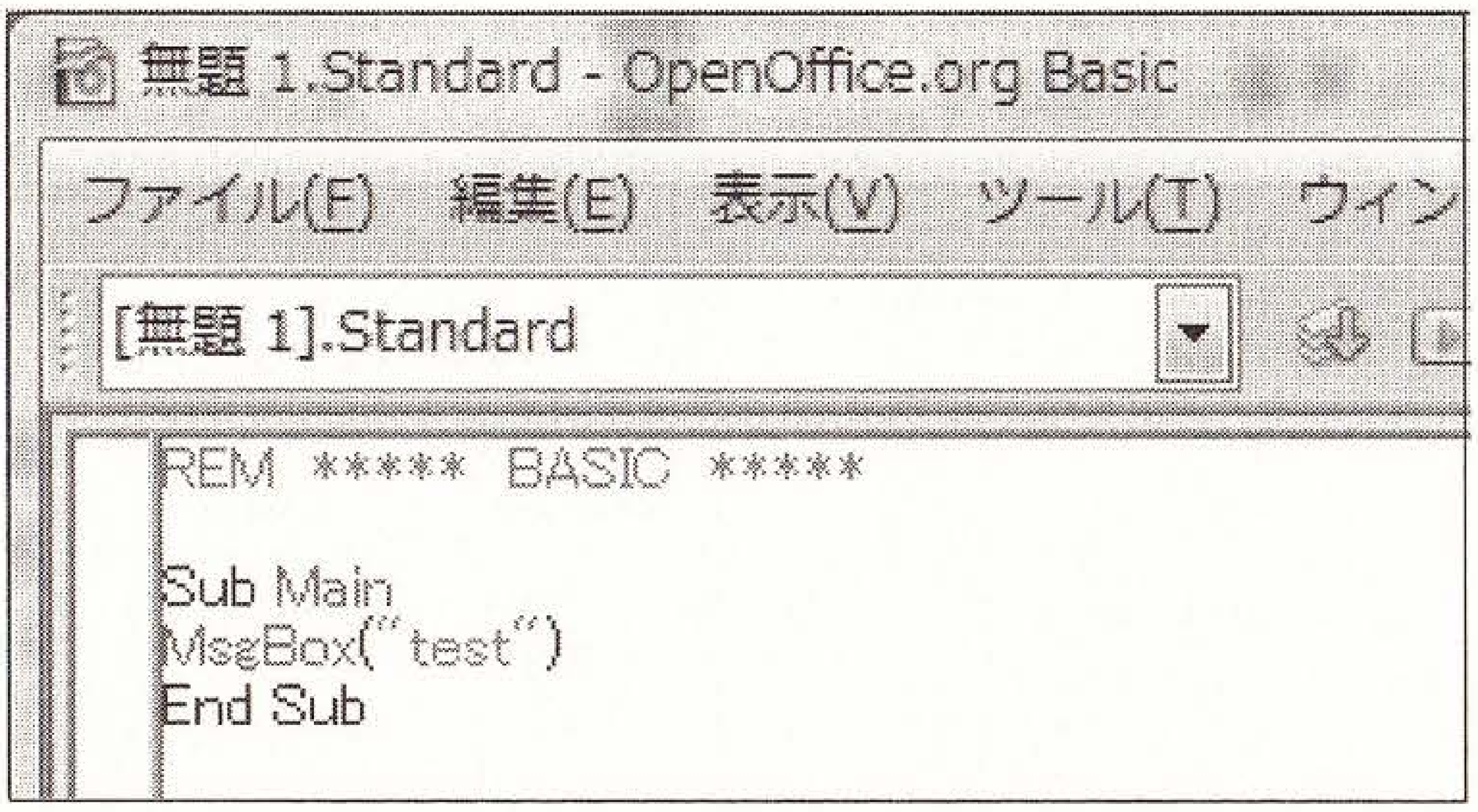
メッセージを表示するには、**「MsgBox( )」**というあらかじめ用意されているサブルーチンを使います。この「MsgBox」は、表示したい文字列を指定すると、その文字列をダイアログ・ボックスで表示します。

では、「Main」サブルーチンを以下のように書き換えてください。

```
Sub Main
MsgBox("test")
End Sub
```

サブルーチンを呼び出すときは、サブルーチンの名前に「()」をつけ、その「()」の中にサブルーチンに指定する**「引数」**(サブルーチンに渡すパラメータ)を記述します。
(今回の場合、「( )」はなくてもいいのですが、分かりやすくするため、つけることにします)。

この例では、「Main」サブルーチンの中で、「MsgBox」というサブルーチンに「"test"」という引数を渡して呼び出しています。



書き換え例

作ったマクロを実行するには、上のほうにある「BASICプログラムの実行」ボタンをクリックします。クリックすると、「test」というメッセージが表示されます。





「BASICプログラムの実行」ボタンをクリックして、マクロが正しく動くことを確認する

1-3 ユーザー側からの入力内容を処理するマクロ

マクロでメッセージ表示ができるようになったので、次はユーザーからの入力をマクロで処理してみましょう。

■ 関数

ユーザーからの入力を受け取るには、「**InputBox( )**」という「**関数**」を使います。

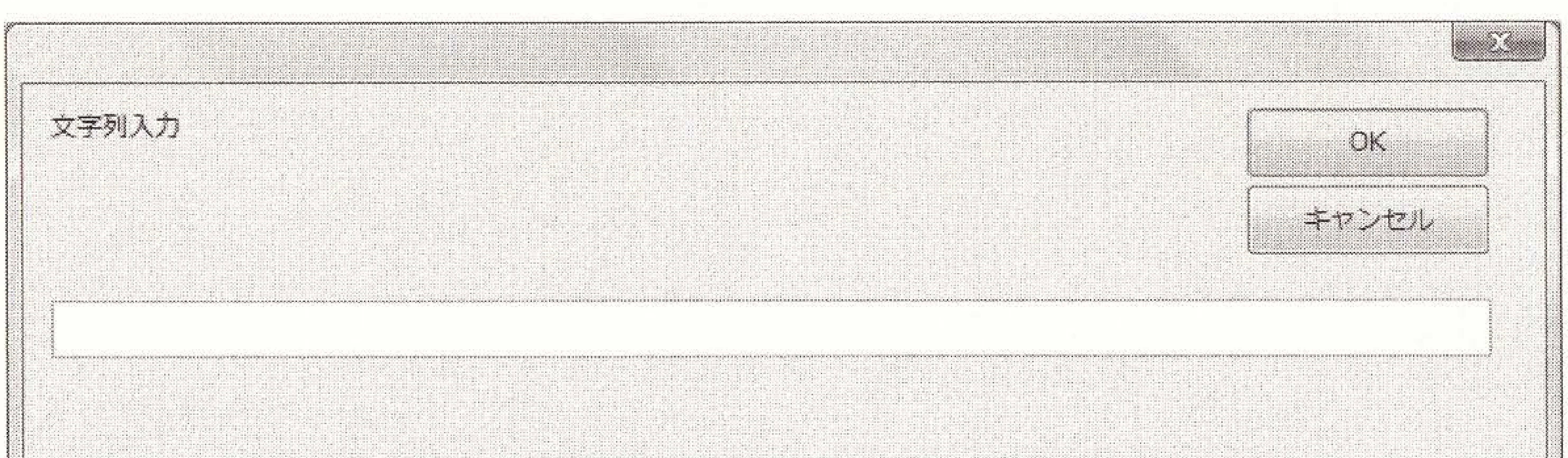
「関数」とは、何かしらの結果(値)を返すサブルーチンのことです。

「InputBox( )」の場合は、呼び出すと入力欄をもつダイアログ・ボックスが表示され、呼び出した結果として入力された文字列が返ってきます。

まず、「Main」サブルーチンを以下のように書き換えて実行してみてください。

```
Sub Main
MsgBox(InputBox("文字列入力"))
End Sub
```

実行すると、最初の引数で指定した「文字列入力」というメッセージと入力欄のあるダイアログ・ボックスが出てきました。これが、「InputBox( )」で作られたダイアログ・ボックスです。



実行結果。引数で指定したメッセージと入力欄が現われる

次に、入力欄に適当な文字列を入れたら、「OK」をクリックしてみましょう。

すると、こんどは先ほど入力した文字列がメッセージ・ボックスに表示されました。これは、「InputBox( )」が返した入力文字列が「MsgBox( )」に渡され、表示されたからです。



**入力欄に適当な文章を入力して「OK」をクリックすると、
入力した内容が新しいダイアログ・ボックスに表示される**

＊

「InputBox( )」は、もう一つ引数を追加して、「入力欄に最初から入っている文字列」を指定することもできます。

サブルーチンに引数を複数指定する場合は、間を「カンマ」(,)で区切ることになっているので、たとえば、

```
Sub Main
MsgBox(InputBox("文字列入力","初期文字列"))
End Sub
```

と書き換えて実行してみると、入力欄に最初から「初期文字列」という文字列が入っているはずです。

1-4 WriterやCalcの画面からマクロを実行する

前項では、「マクロの編集」画面(IDE)から「実行」ボタンをクリックしてマクロを実行しました。

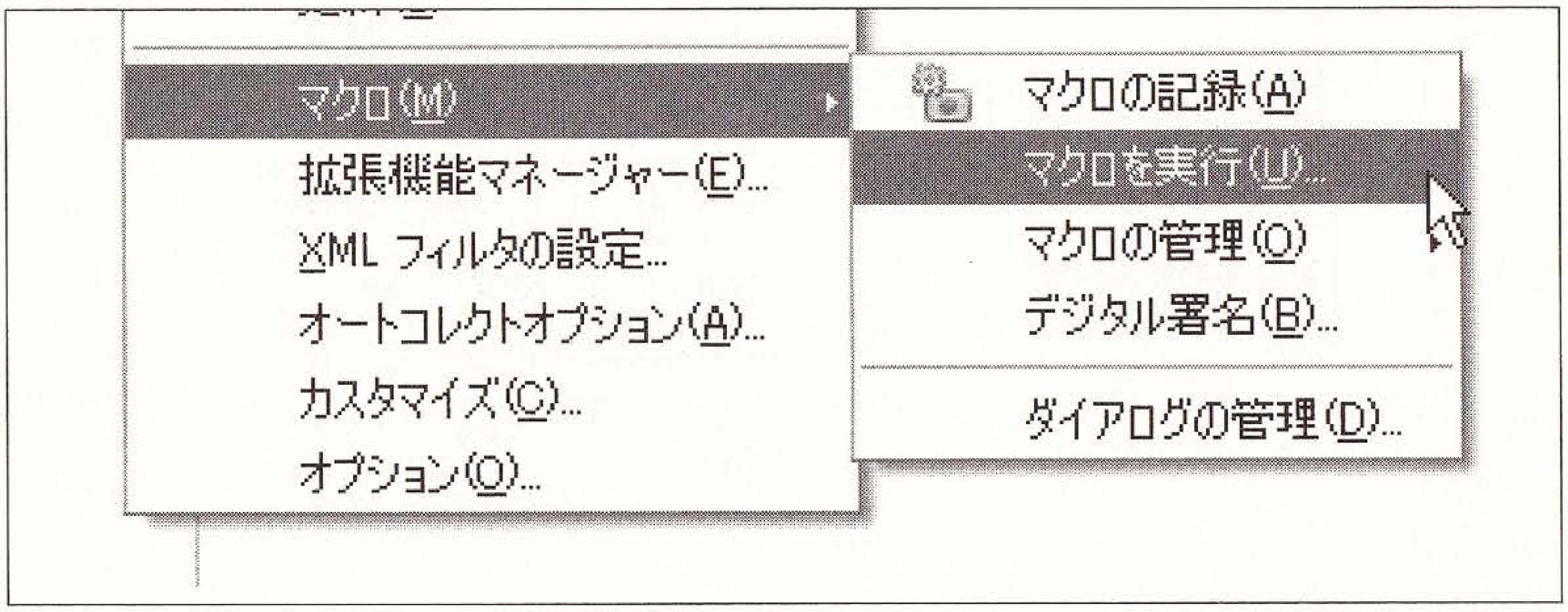
「実行」ボタンをクリックすると、現在編集しているマクロ(モジュール)の「最初のサブルーチン(か関数)」が実行されます。前項では、「Main」という一つのサブルーチンを作ってみたので、「Main」が実行されたわけです。

■ アプリケーションからマクロを実行

ただ、実際にマクロを使う場合は、マクロを編集するためのIDEからではなく、「Calc」のシートなど、アプリケーションの画面から呼び出したいですね。また、複数のマクロのサブルーチンがある場合は、呼び出すサブルーチンを指定する必要もあります。

そうしたマクロのサブルーチンや関数名を指定して実行するのが、「ツール」メニューの「マクロ」内にある「マクロの実行」メニューです。

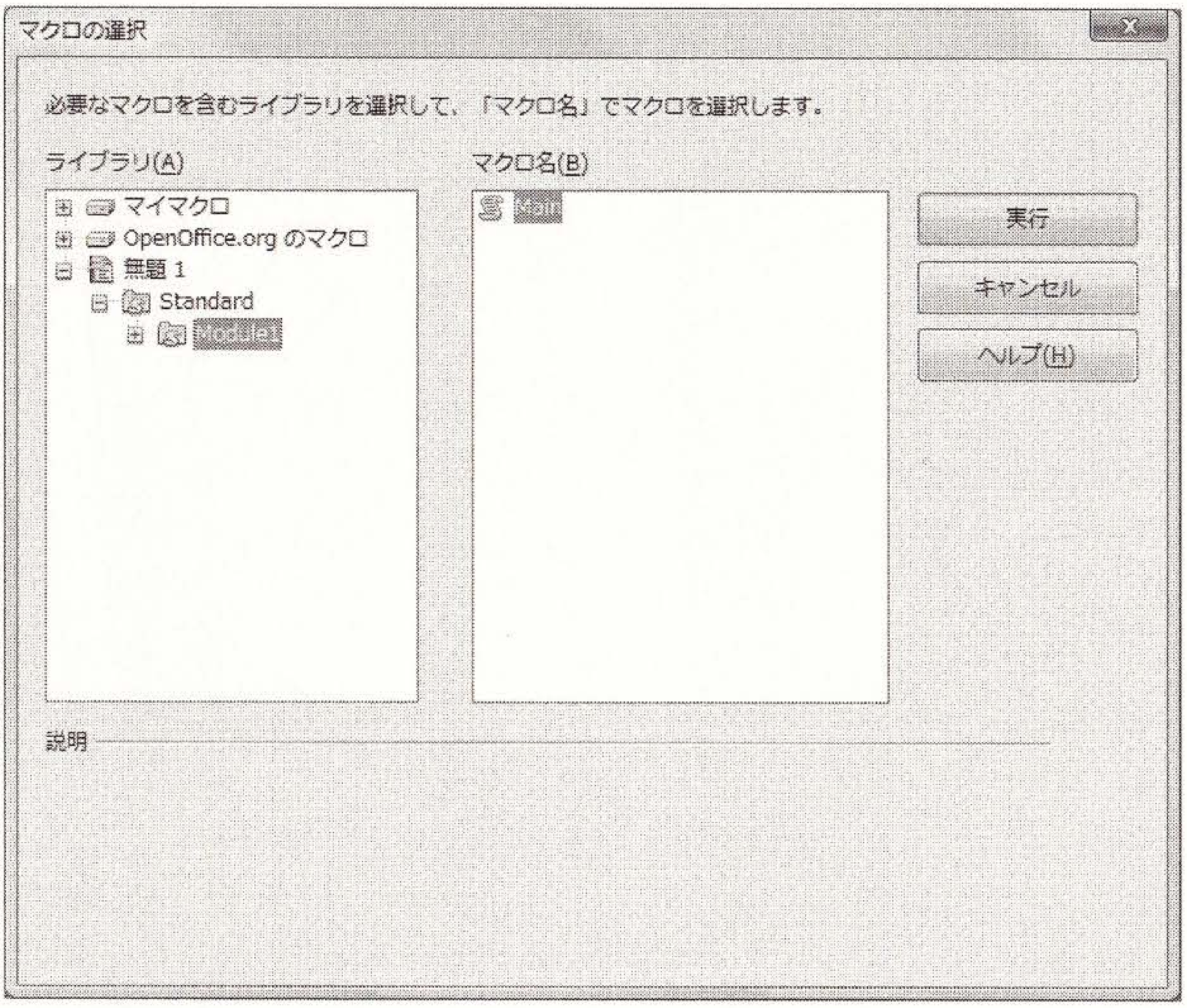
まず、「ツール」→「マクロ」→「マクロの実行」メニューを選択し、実行するマクロの画面を呼び出してみましょう。



「マクロの実行」を選択

「マクロの選択」ダイアログが出てきたら、マクロを作ったときと同じく「無題1」など現在のファイルを選び、**1-2節**で編集した「Module1」をクリックします。

すると、右側にモジュールに含まれる「サブルーチン」「関数」の一覧が表示されるので、「Main」を選んで「実行」してみましょう。



「マクロの選択」ダイアログ
ライブラリは「Module1」、マクロ名は「Main」を選択して、「実行」をクリック。

「Main」サブルーチンが実行され、**1-2節**と同じく「test」というメッセージが表示されるはずです。

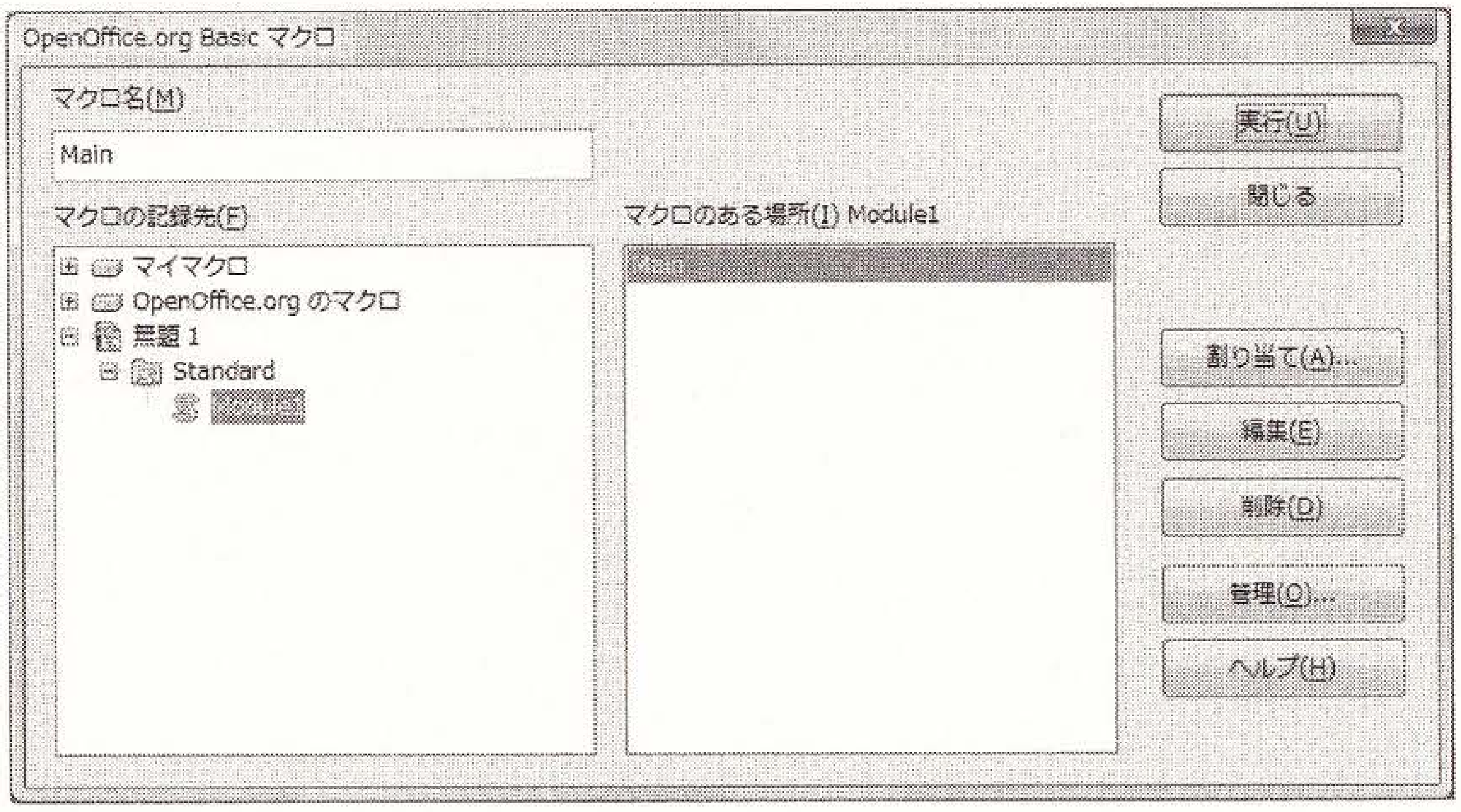


1-2節と同じ結果が返ってくる

■ 複数のサブルーチンを使い分ける

次に、もう一つサブルーチンを追加し、追加したサブルーチンをメニューから実行してみましょう。

「ツール」→「マクロ」→「マクロの管理」→「OpenOffice.org BASIC」からマクロの管理画面を表示させたら、**1-2節**で作った「Module1」を選択し、「編集」ボタンをクリックしてIDE画面に入ってください。



1-2節で作ったマクロを編集し直す

続いて、「Main」サブルーチンの下に、以下のような「test」サブルーチンを追加します。

```
Sub test
	' メッセージ表示
	MsgBox("test2")
End Sub
```

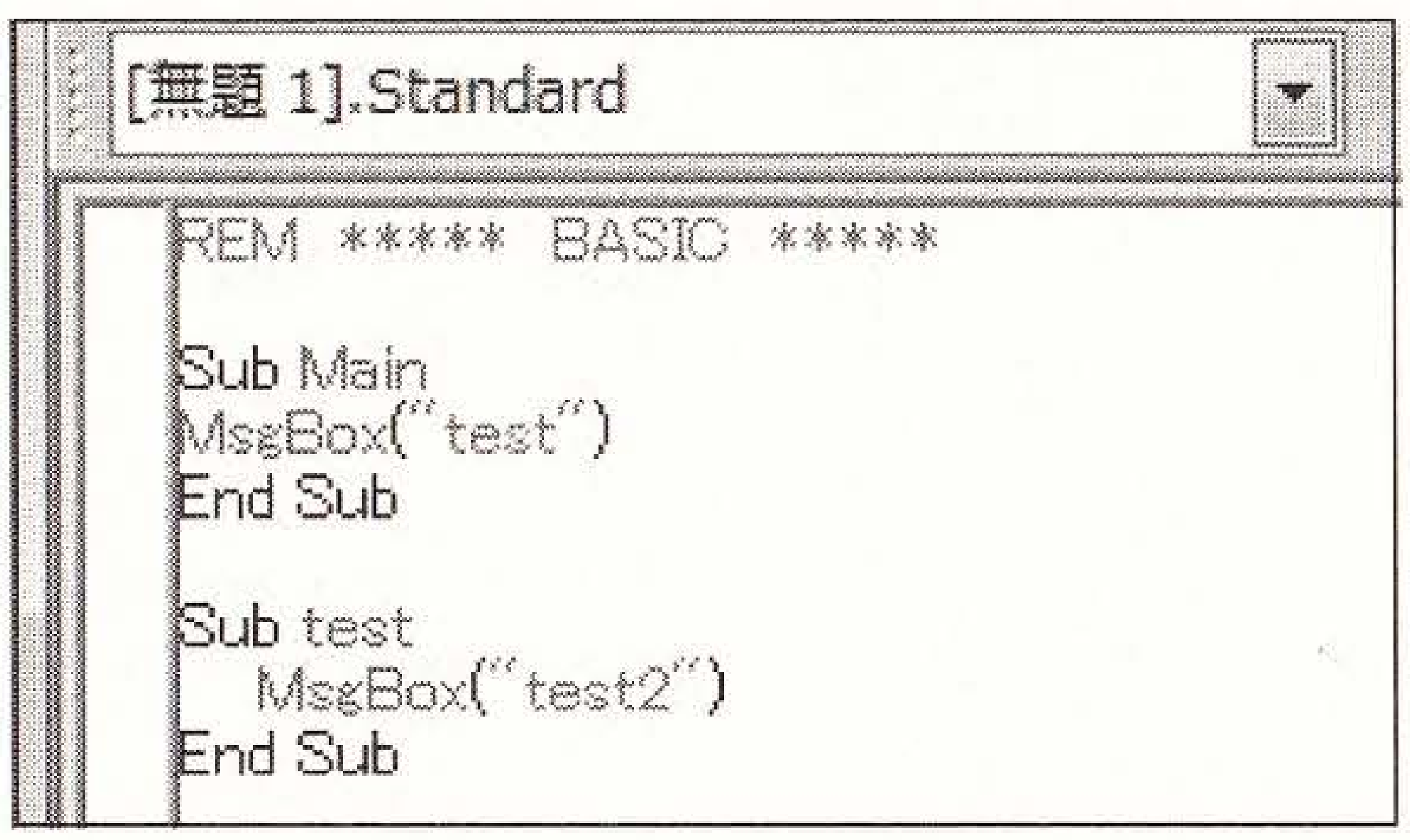
今回は、「MsgBox()」の前が空いていますが、これは読みやすいように「**Tab文字**」を入れてあるからです。

「Tab文字」は入力しなくてもかまいませんが、こうして字下げをすると読みやすくなるので、なるべく適宜「Tab文字」を入れて、読みやすいマクロを作るようにしましょう。

2行目の「'」は、マクロにコメント(説明)を入れるものです。「'」以降の改行までの部分は、マクロ実行時には実行されません。

コメントにマクロの変数や処理に関する説明を書いておくと、後で修正するときや他の人がマクロを使うときに役に立つでしょう。

また、行の先頭に「'」を付けると、その行を実行しないようにできる(コメントアウト)ので、テストや修正の際に一部の処理を無効化するために使うこともできます。

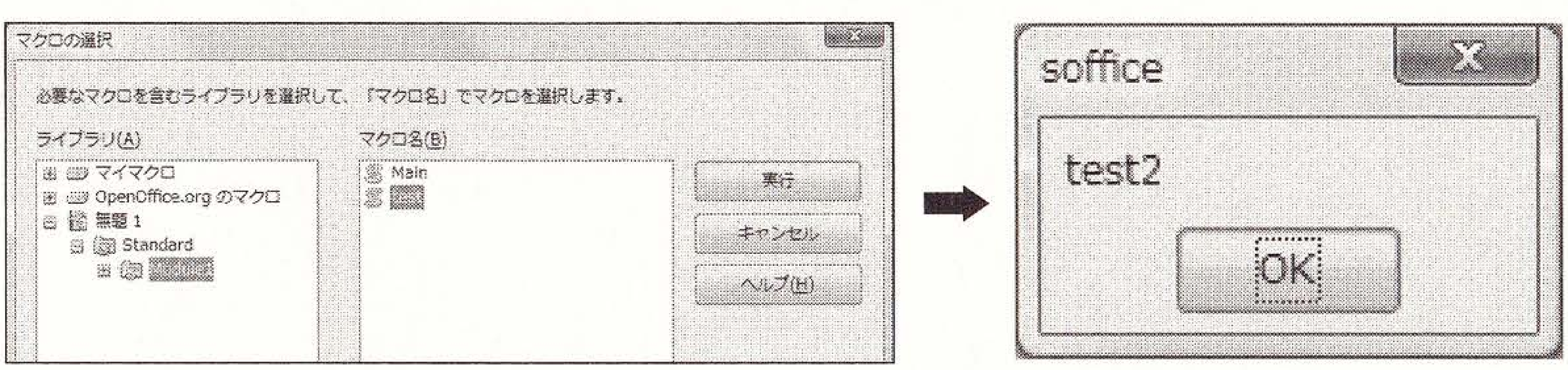


Tabを入れると構造が分かりやすくなるので、できるだけ入れておくことをお勧めする。

さて、これで「test2」というメッセージを表示する「test」サブルーチンができたので、一度IDEを閉じてアプリケーションの画面に戻りましょう。

そして、先ほどと同じく、メニューから「マクロの選択」ダイアログを表示させると、先ほど追加した「test」サブルーチンがあります。

選択して実行すると、「test2」という文字列が表示されました。



ライブラリは「Module1」、マクロ名は「test」を選択し、実行をクリックすると、「test2」という文字列ダイアログが表示される

＊

以上で、マクロのサブルーチンを作ってメニューから呼び出し実行する一連の流れは掴めたと思います。

第2章

マクロ・プログラミングの基礎

「マクロ・プログラミング」では、加算や減算などの簡単な演算から、関数や配列を使った複雑な計算まで、実にさまざまな処理が行なえます。

この章では、「マクロ・プログラミング」で使える処理について、そのテクニックを学んでいきます。

2-1 変数

本格的なマクロを作るために欠かせないのが、**「変数」**です。
「変数」とは、データを記憶しておく記憶領域のことで、適当な名前(変数名)をつけて、その変数名で変数が記憶しているデータを参照します。

変数は、マクロの中で計算結果を一時的に記憶しておいたり、一定回数の処理を繰り返すときに何回処理を繰り返したかを数える「カウンター」として利用したり、ユーザーの操作で得られる情報を受け取ったりと、さまざまな場面で利用されます。

変数を利用するには、**「Dim文」**で変数の「型」(格納する情報の種類)を指定し、変数を宣言します。変数の型には多くの種類がありますが、とりあえずはよく使われる、

Integer	……整数(-32768～32767)
Long	………整数(±約21億)
Double	……実数(小数点以下を含む数値)
String	………文字列

といった型に慣れておきましょう。

変数の宣言は、

```
Dim 変数名 As 変数の型
```

という形で行ないます。

変数を宣言したら、「=」で値を代入(設定)します。たとえば、

```
Dim str As String
```

として文字列型変数「str」を宣言したら、

```
str="文字列"
```

のようにして変数「str」に格納する文字列を設定するわけです。

数値型の変数も同様に、

```
Dim i As Integer

i=123
```

などとして、変数の宣言や代入を行ないます。

さらに、変数には、

```
str=InputBox("文字列を入力")
```

のように、関数が返す値を代入することもできます。

2-2 変数を使った計算

変数の宣言や代入を試してみたので、こんどは変数を使って計算をしてみましょう。

■ 演算子

「OOo」のマクロでも、通常の加減乗除といった演算ができます。

演算は、**「演算子」**と呼ばれる記号を使い、演算子ごとに「乗除算は加減算より優先」といった優先順位があるのも、通常の数式と同じです。また、「( )」で優先順位を指定することもできます。

●主な演算子

+	……………加算
-	……………減算
*	……………乗算
/	……………除算
^	……………累乗
mod	………除算の余り

●数式の例

```
Dim result As Long

result = 1 + 2 + 3 ……resultの値は6

result = 1 + 2 * 3 ……resultの値は7

result = (1 + 2) * 3……resultの値は9

result = 2 ^ 4 …………resultの値は16

result = 17 mod 4 ……resultの値は1

result = result * 2 ……resultの値が2倍になる
```

「MsgBox( )」を使うと、数値を文字列として表示できるので、適当な変数を宣言し、数式の結果を代入してから表示してみましょう。

マクロの管理画面から「OpenOffce.org Basic」の編集画面を呼び出し、以下のマクロを入力、実行してみてください。

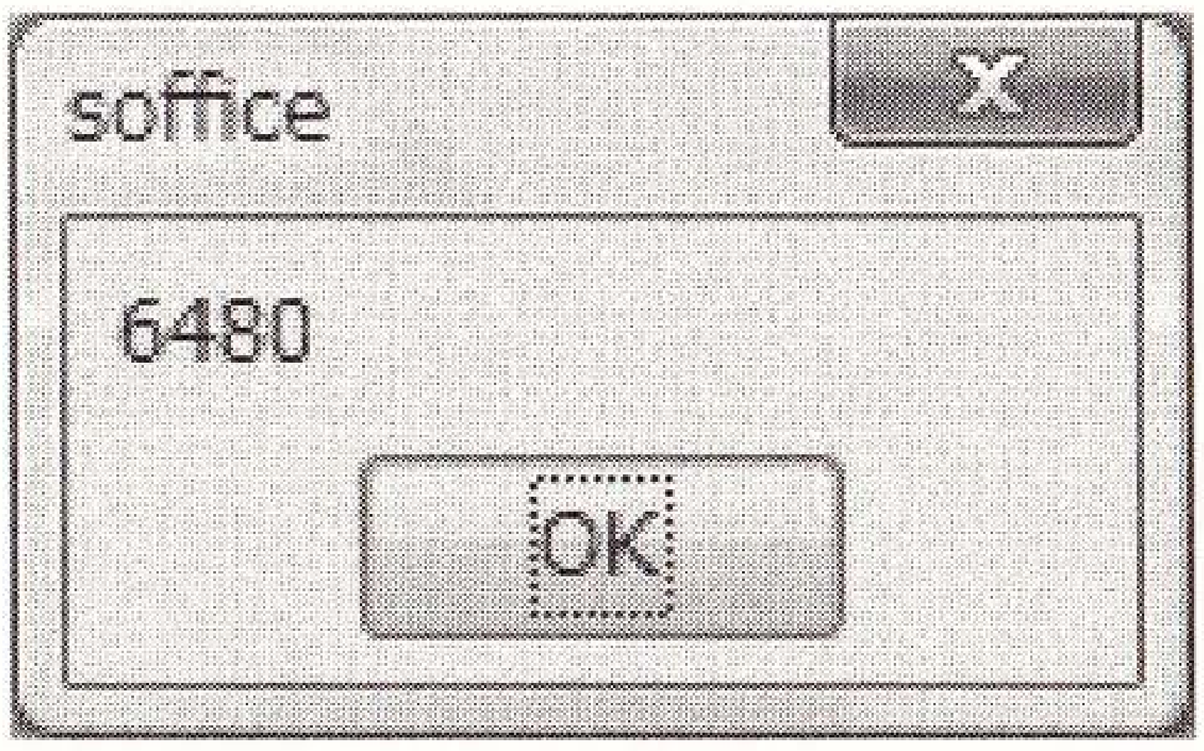
```
Sub Main

    Dim result As Long

    result = ((1 + 2 + 3) ^ 4) * 5

    MsgBox(result)

End Sub
```



実行結果

計算結果を確認したら、式の部分をいろいろ書き換えて、思ったとおりの結果になるか確かめてみてください。

演算子の「優先順位」については、式が複雑になると分かりにくくなるので、「()」で先に行なう演算を強調し、分かりやすい式を書くようにしましょう。

なお、数値の計算を行なう際、特に累乗の計算が入る場合は、計算結果が変数に格納できる数値の範囲を超えないか、注意が必要です。「Long型」変数には「±21億」程度の数字が格納できますが、この範囲を超える可能性がある計算を行なう場合は、「Double型」の使用も検討してみてください。

■ 16進数での表記

また、「OpenOffice.org BASICマクロ」では、数値を16進数で表記することもできます。数値を16進表記にするには、先頭に「&H」を付けて、

```
&HFF
```

などとしてください。たとえば、

```
Sub Main

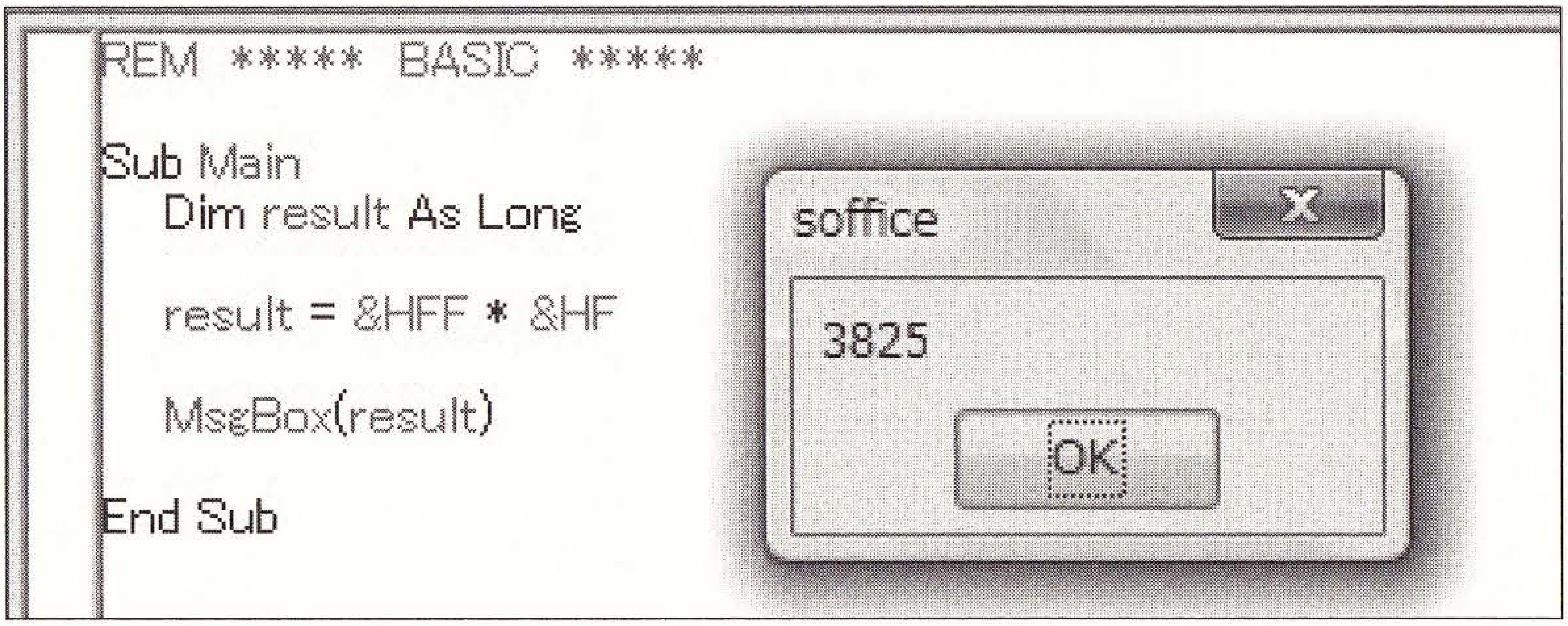
    Dim result As Long

    result = &HFF * &HF

    MsgBox(result)

End Sub
```

とすると、「255*15=3825」と表示されます。



16進数での表記も可能

色の設定などでは、16進表記のほうが分かりやすいので、16進表記にも慣れておくようにしましょう。

2-3 数値と文字列を組み合わせた演算

前項では、演算子を使って「数値の計算」をしてみました。

2つの数値に「+」演算子を使うと「足し算」ができましたが、実は、「+」演算子は、「文字列」に対しても使うことができます。

■ 文字列＋文字列

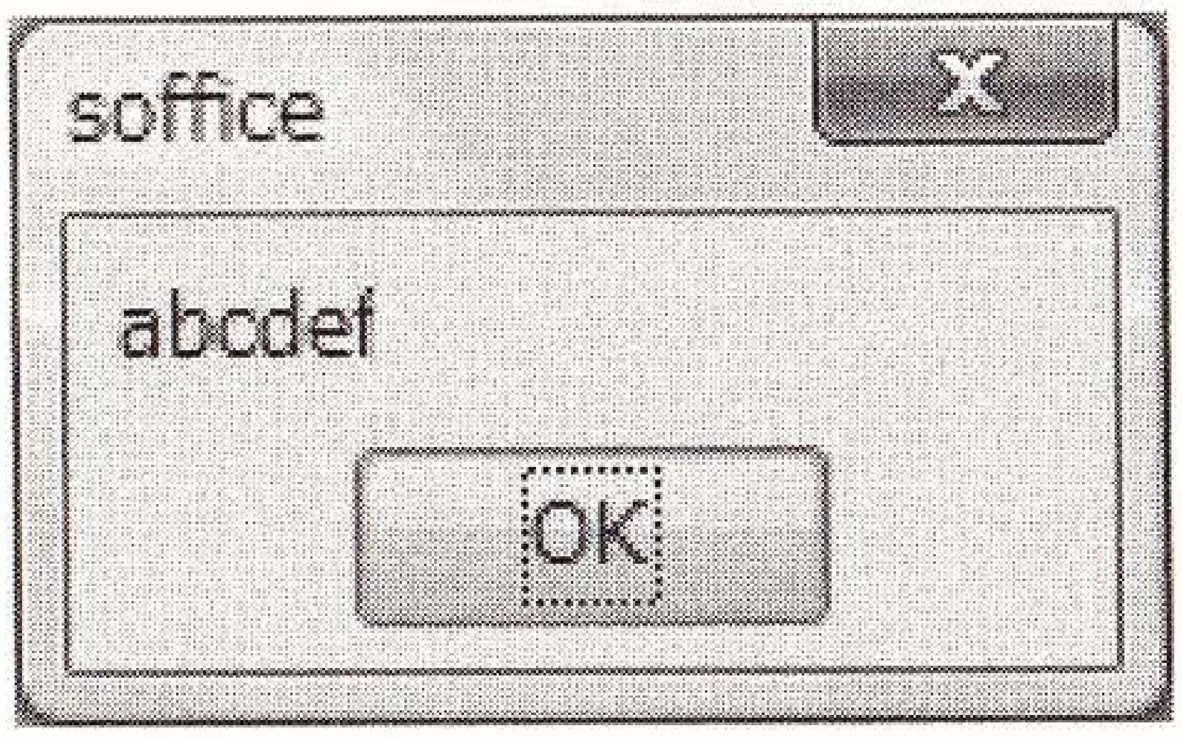
たとえば、以下のマクロを実行してみてください。

```
Sub Main

    MsgBox("abc" + "def")

End Sub
```

実行すると、「"abc"」に「"def"」が足されて「abcdef」という文字列が表示されました。



実行結果

■ 文字列＋数値

次に、以下のマクロはどうでしょう。

```
Sub Main
    Dim num As Long
    num = 123
    MsgBox("abc" + num)
End Sub
```

実行すると、「abc123」というメッセージが表示されました。



実行結果

今回は、「文字列 + 数値型変数」、つまり「文字列 + 数値」という加算でしたが、この場合は「数値」が「文字列」として連結されるわけです。

■ 数値＋文字列

ただし、逆に「数値＋文字列」という演算は、何の効果もありません。

この場合、文字列の部分が数値としては「0」と評価されるので、数値が変わらないのです。

試しに、

```
Sub Main

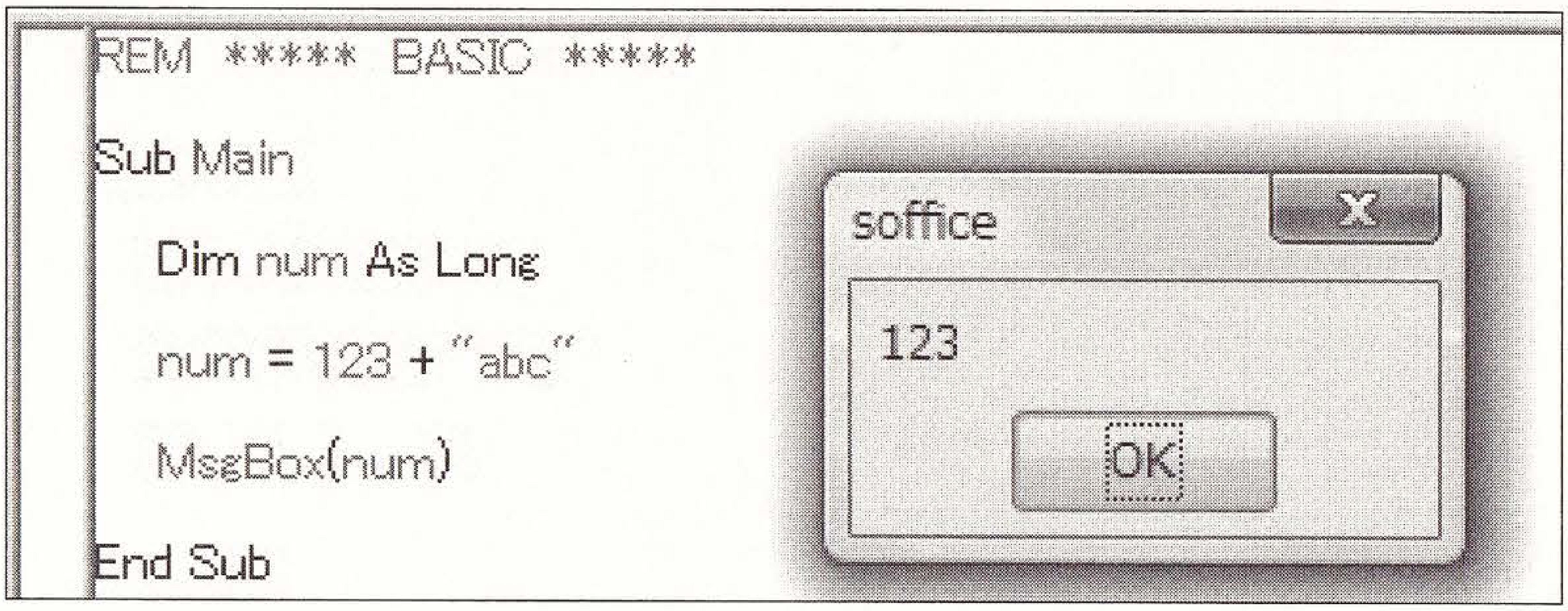
    Dim num As Long

    num = 123 + "abc"

    MsgBox(num)

End Sub
```

というマクロを実行すると、「123」というメッセージが表示されます。



「数値+文字列」という順で演算を行なっても、「数値」のみの結果となる

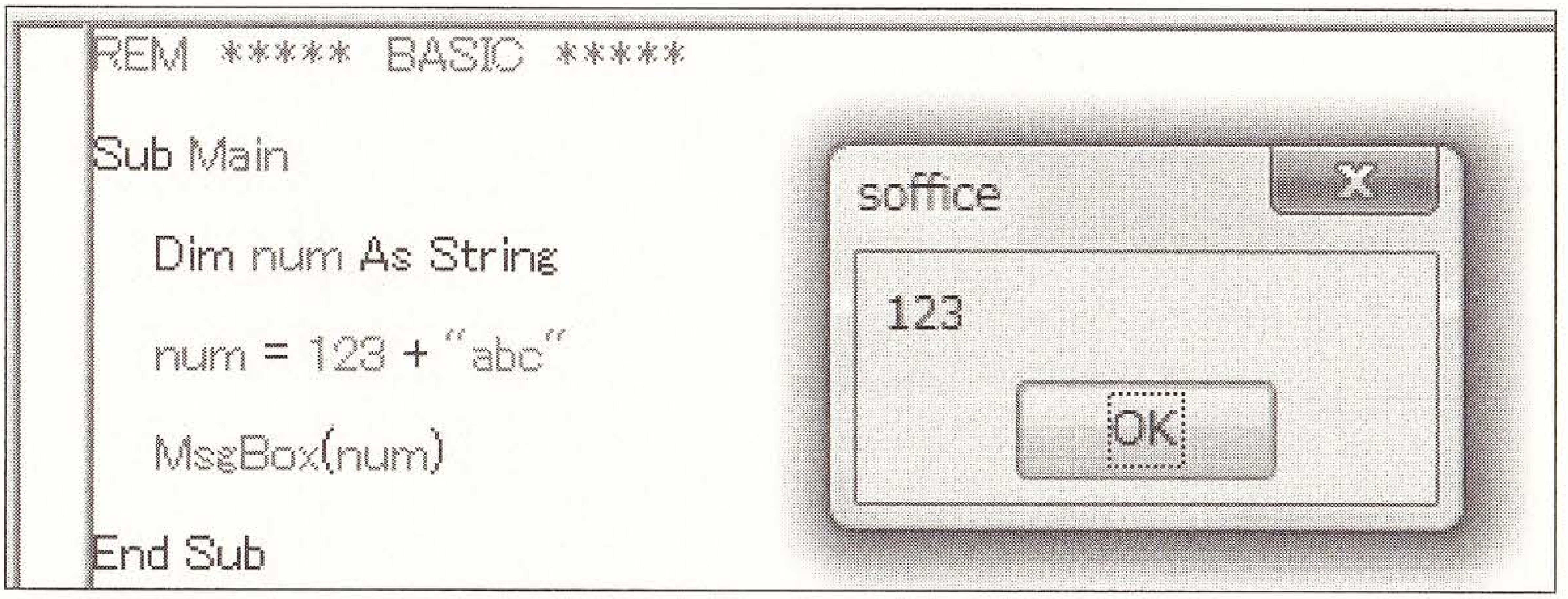
なお、このマクロの場合は変数「num」を文字列型として、

```
Sub Main
    Dim num As String
    num = 123 + "abc"
    MsgBox(num)
End Sub
```

としても、同じ結果になります。それは、

```
num = 123 + "abc"
```

という式が、まず「123 + "abc"」という数値を計算する式として演算が行なわれ、その結果の数値が文字列に変換されるからです。



文字列として表記しても、結果は同じ

■ 型の自動変換

以上のことをまとめると、

「文字列」に「数値」を「＋」すると、「数値」が「文字列」として加えられる。 「数値」に「文字列」を加えても、式の値は「数値」そのままで、変わらない。

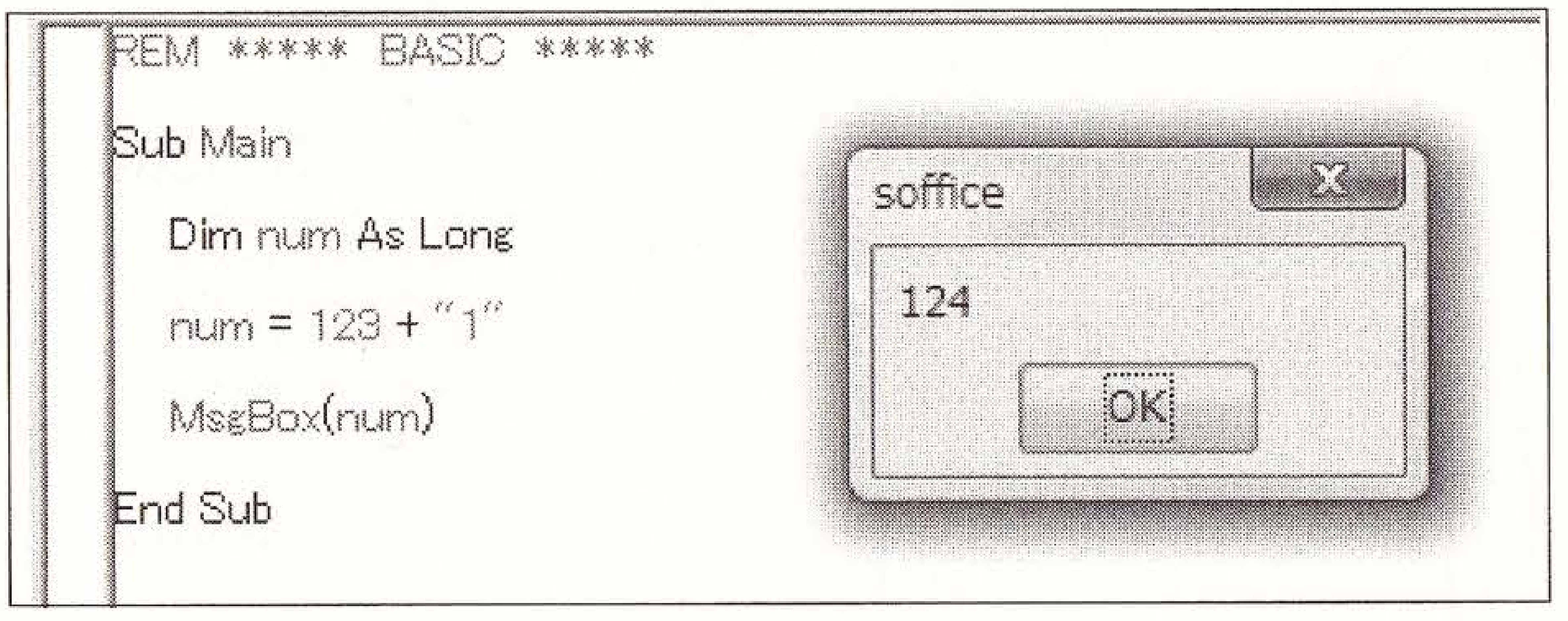
ということになるのですが、ややこしいことに、数値に「数値として意味をもつ文字列」を加えると、その結果は「数値＋文字列が表わす数値」になってしまうのです。

たとえば、

```
Sub Main
    Dim num As Long
    num = 123 + "1"
    MsgBox(num)
End Sub
```

というマクロを実行すると、「124」というメッセージが表示されます。

これは、「"1"」という文字列が、自動的に「1」という数値として扱われるためです。



結果は「123+1」と同じになる

こうしたややこしい現象が起こる原因は、「型の自動変換」という働きのためです。「OpenOffice.org Basic」では、今回のように文字列や数値などが文脈によって、自動的に別の型として扱われることがあるのです。

具体的に、どういったときにどんな型変換が起こるかは、少しずつ慣れてくれば感覚的に分かってくるのですが、「変数の型」に注意しないと思わぬ結果になることもあるかもしれません。

■「&」を使った連結

最後に、文字列の連結についてもう一点。

実は、文字列の連結は「&」でも行なうことができて、この「&」を使った場合、「数値&文字列」の結果が、「数値+文字列」の結果と違ったりします。

```
Sub Main

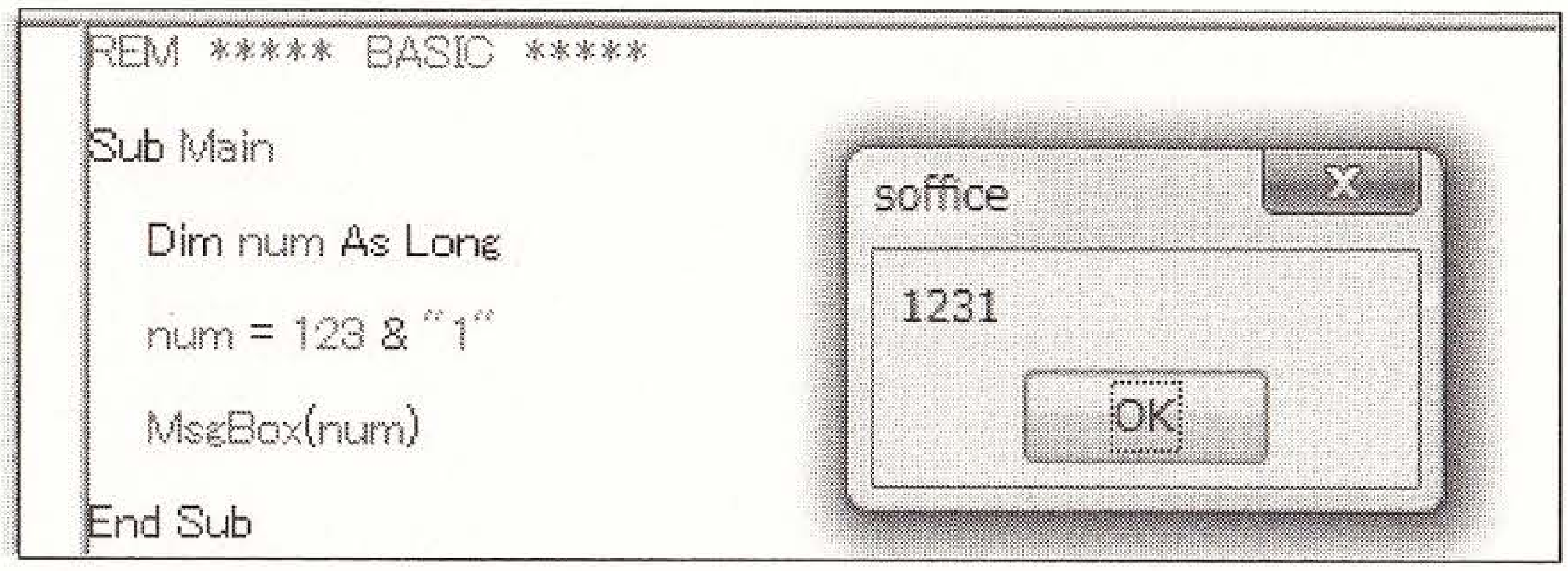
    Dim num As Long

    num = 123 & "1"

    MsgBox(num)

End Sub
```

というマクロを実行すると、こんどは「1231」になります。



結果は「"123"+"1"」と同じになる

「123+1」では、「"1"」が数値として扱われましたが、こんどは「123」という数値のほうが「文字列」として扱われ、連結結果の文字列「1231」が、変数「num」に代入されたようです。

「+」の場合は、(文字列であっても)なるべく数値として解釈しようとし、「&」の場合は、(数値であっても)なるべく文字列として解釈しようとする、と言えるかもしれません。

「&」のほうが、「文字列の連結」という面では本来の意味に近いでしょうか。

＊

この文字列と数値の間の変換は、思わぬ結果を生むこともあるので、「数値として入力された」「読み込んだ文字列」の扱いには、注意しましょう。

2-4 関数

基本的な演算ができるようになったので、次に、独自の**「関数」**を作ってみましょう。

1章で説明しましたが、「関数」とは、「値を返すサブルーチン」のことです。

「OOo」のマクロには、あらかじめ多くの「サブルーチン」や「関数」が用意され、マクロから呼び出すことができますが、このサブルーチンや関数は自分で定義することもできます。

「複雑な演算処理」や「入出力の検査」など、何度も使いまわす処理をサブルーチンや関数にまとめておけば、マクロをすっきりと書くことができるでしょう。また、一度作ったサブルーチンや関数を、他のマクロで使いまわすなど、プログラミングの“資産”として活用しやすくなります。

■関数の定義

関数を定義するには、以下のようにします。

```
Function 関数名(引数) As 関数の型

     処理内容

     関数名=関数の結果

End Function
```

引数の指定は、

```
変数名 As 変数の型
```

という形で行ないます。

たとえば、「引数として渡されたLong型変数の値に1を加えた値を返す関数inc()」を定義するなら、

```
Function inc(val As Long) As Long
    Dim result As Long
    result = val + 1
    inc = result
End Function
```

といった感じで書くわけです。
(単に「1」を加える関数を作るなら、「inc = val +1」とするのが普通ですが…)。

こうして定義した関数は、あらかじめ用意されている関数と同じく、

```
関数名(引数)
```

として呼び出すと、結果が返ってきます。

■ 値渡し

ただし、こうして渡された変数を、関数の中で書き換えると、呼び出した側にも影響が及ぶ(関数に渡した変数の値が変わってしまう)ので、注意してください。

たとえば、以下のマクロを実行してみてください。

```
Sub Main
    Dim i As Long
    i = 1
    MsgBox "i=" & i
    test(i)
    MsgBox "i=" & i
End Sub
Function test(arg As Long) As Long
    arg = 2
    test = 1
End Function
```

このマクロでは、まず「Main()」で変数「i」を用意し、「i」に「1」を代入します。

そして、その変数を「test( )」関数に渡しているのですが、「test( )」を呼び出す前後で「i」の値を表示してみると、「Main( )」内で変数「i」には一切手を加えていないのに、「i」の値が「2」に変わっています。

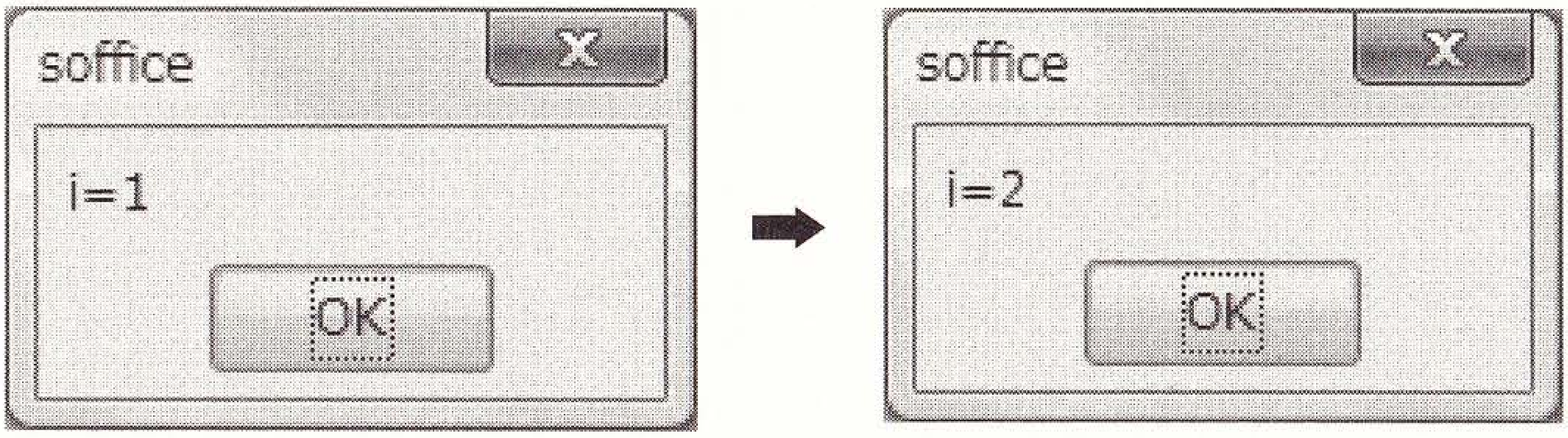
これは、「test()」内で渡された変数「arg」に、「2」を代入した結果です。

関数やサブルーチンを呼び出す際に引数として変数を渡すと、その変数自身(変数の参照)が関数やサブルーチンに渡されます。

```
Sub Main
  Dim i As Long
  test(i)
End Function
```

```
Function test(arg As Long) As Long
  ..
End Function
```

という形で、「Main( )」から「test( )」に変数「i」を渡すと、「Main( )」内の変数「i」と「test( )」で引数として受け取る変数「arg」が“同じもの”として扱われるわけです。このため、「test( )」内部で「arg」の値を変更すると、その影響は「Main()」の変数「i」にも及びます。



関数の中で、渡された変数の値を変更すると、
関数に渡された変数の値も変わってしまう

こうしたことを避けるために、「OpenOffice.org BASIC」には、呼び出し側の指定した値をコピーして、関数内で使わせるようにする**「値渡し」**という仕組みが用意されています。これは、引数の宣言時に**「ByVal」**を付けることで、変数自身ではなく、その値「1」をコピーして、引数とするものです。

＊

たとえば、「test()」を以下のように書き換えてみましょう。

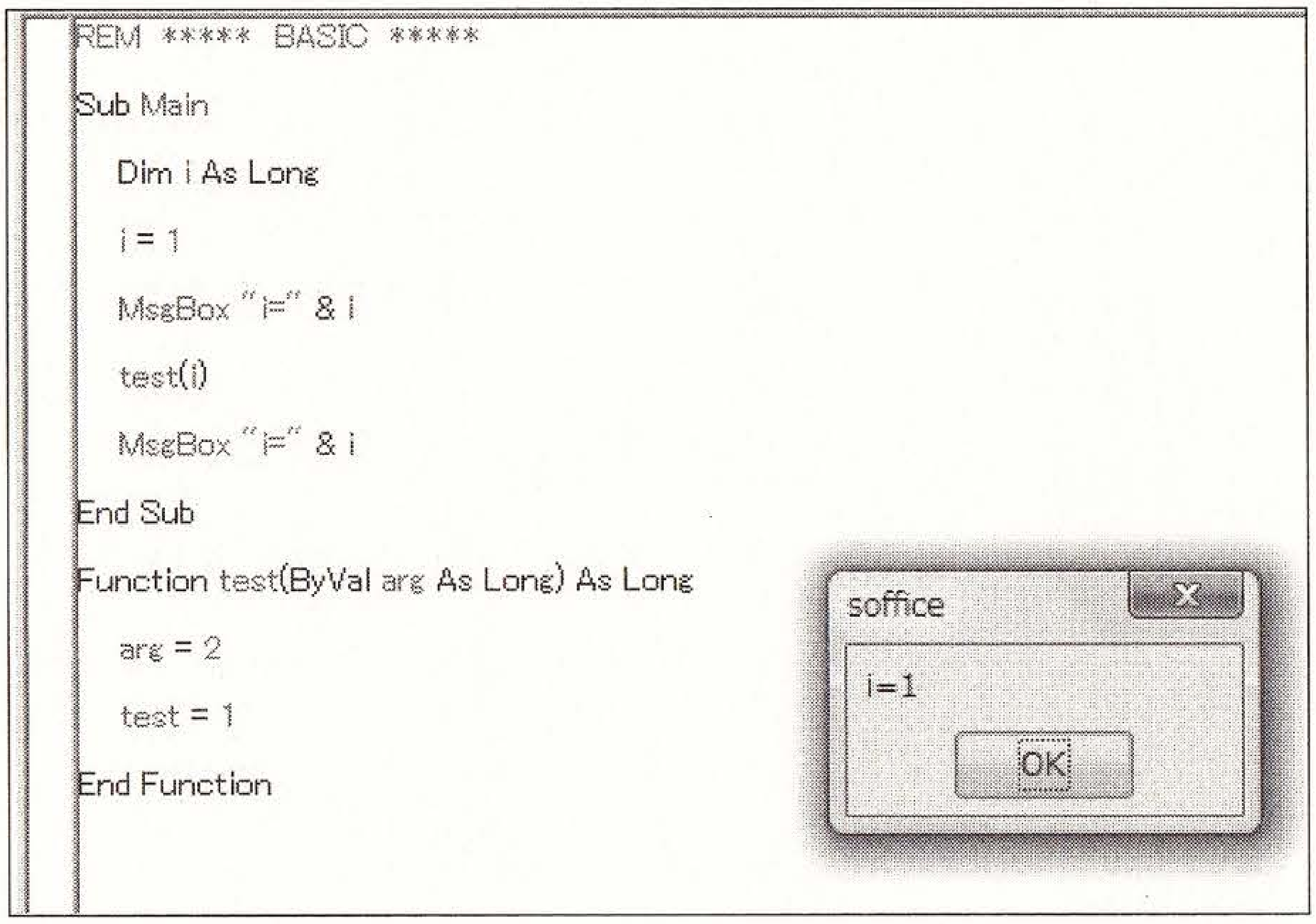
```
Function test(ByVal arg As Long) As Long

    arg = 2

    test = 1

End Function
```

実行すると、「test( )」呼び出しの前後で、値が変わらなくなっていることが確認できるはずです。



値渡しだと、変数の値は変わらない

2-5 条件分岐

マクロを書いていると、「**条件による分岐**」を行ないたくなることがあります。

たとえば、「ある変数の値が5未満ならこの処理、そうでなければこの処理」と変数の値によって処理を振り分けたり、「式の値がある変数より大きければ、特定のサブルーチンを呼び出す」といった状況です。

■ If文

「OpenOffice.org Basic」には、こうした条件分岐のために「**If文**」という制御構文が用意されています。

この「If文」は、

```
If （条件） Then

    条件を満たしたときの処理

End If
```

という構文で、指定した条件を満たすと、「Then」から「End If」の間の処理を実行します。

また、「Else」を指定すると、「条件を満たさない場合」にも特定の処理を行なえます。

```
If (条件) Then
    条件を満たしたときの処理
Else
    条件を満たさなかったときの処理
End If
```

「If文」で指定する条件は、「Bool型の値をもつ式」です。

「Bool型」とは、「True」(真)か「False」(偽)か、という2種類の値のいずれかとなる型で、「True」なら条件を満たしたものとして、「False」なら満たさなかったものとして扱われます。

■比較演算子

「If文」に条件として渡す式には、等号や不等号などの**「比較演算子」**による「比較式」や、「Bool型」の値を返す関数を利用します。

比較演算子には、

=	……………等しい
<>	…………等しくない
<	……………左側が小さい
>	……………右側が小さい
<=	…………左側が右側より小さいまたは等しい
>=	…………左側が右側より大きいまたは等しい

といったものがあり、たとえば、

```
Sub Main

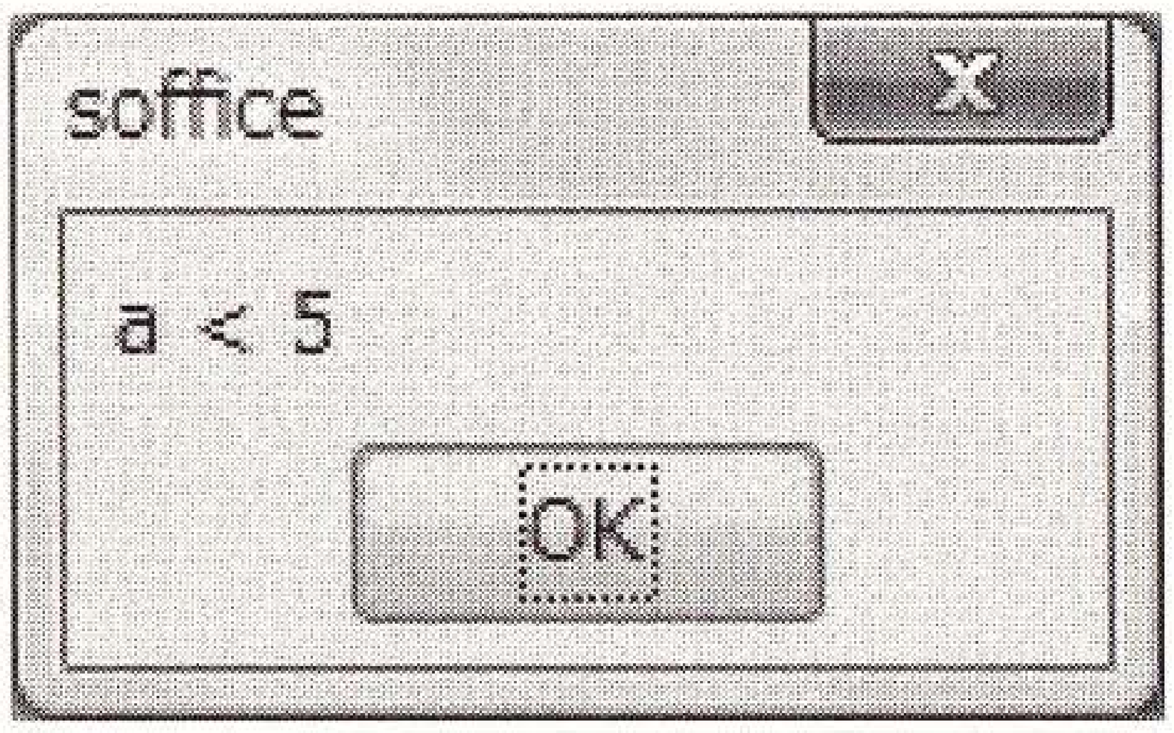
    Dim a As Long

    a = 3

    If (a < 5) Then
        MsgBox("a < 5")
    End If

End Sub
```

というマクロを実行すると、「a」には「3」が代入され、「a < 5」(変数aは5より小さい)という比較は真になるので、「If文」の中の処理が実行され、「a < 5」という文字列が「MsgBox()」で表示されます。



「a」が「5以上」の場合は、実行しても何も起こらない

＊

最後に、先のマクロを以下のように書き換えて実行してみましょう。

```
Sub Main

    Dim a As Long

    a = 3

    If (1) Then
        MsgBox("a < 5")
```

```
    End If

End Sub
```

「If文」に指定する条件が「数値」になっていますが、実行してみると先ほどと同じくメッセージが表示されました。つまり、「If文」の条件を満たした、として扱われたわけです。

次に、「If文」の「( )」の中を、「1」から「0」に書き換えてみると、こんどは何も表示されません。つまり、「偽」と判定されたわけです。

実は、「If文」などの条件式に数値を渡すと、「0なら偽、それ以外は真」として扱われます。この性質を使うと、ある数値型変数「val」の値が「0」でなければある処理を行なう、という「If文」を、

```
If (val) Then
    Func( )が0以外の値を返したときの処理
End If
```

などと書くこともできます。

ただし、こうした書き方は「分かりにくい」ので、あまり使わないようにしましょう。

2-6 ループ処理①

「If文」で条件分岐ができるようになったので、次は**「条件を満たすまで処理を繰り返す」(ループ処理)**ようなマクロを試してみましょう。

ループ構造を使えるようになると、「条件が満たされている間、ループ処理を繰り返す」ことができるようになります。

■「Do～Loop」構文

「OpenOffice.org Basic」には、継続(終了)条件を指定するループのための構文として、**「Do～Loop」**という構文があります。

これは、

```
Do
    ループ処理
Loop While 継続条件
```

といった形で「継続条件を満たす間、ループ処理を実行」したり、

```
Do
    ループ処理
Loop Until 終了条件
```

のように、「終了条件が満たされるまで、ループ処理を実行」したりする構文です。

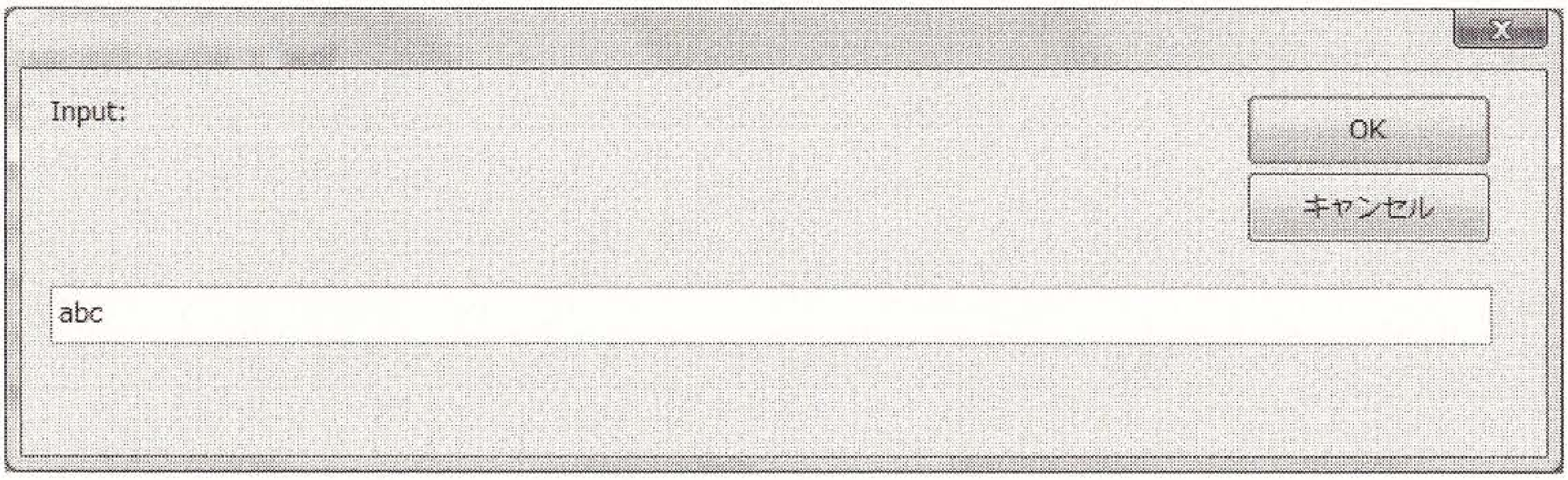
たとえば、以下のマクロを実行してみてください。

```
Sub Main
    Dim inputStr As String
```

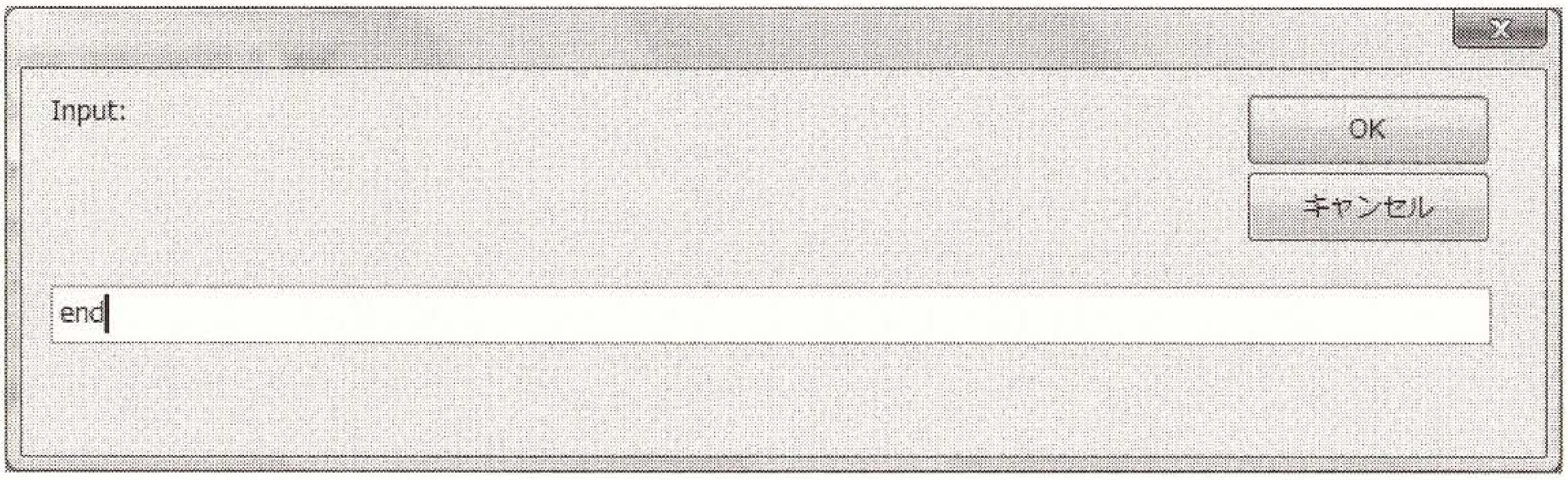
```
    Do
        inputStr = InputBox("Input:")
    Loop Until inputStr = "end"
    MsgBox("ループ終了")
End Sub
```

入力ダイアログに「abc」などと入れると、終了条件を満たさないためにループ先頭に戻り、また入力ダイアログが出てきます。

何度か適当な文字列を入れたら「end」と入れてみましょう。こんどは、ループ終了条件の「inputStr = "end"」が満たされてループが終了し、「ループ終了」というメッセージが出てきます。



↓



↓



終了条件を満たせば(「end」と入力すれば)、ループが終了する

■「While～Wend」構文

上の例では、まずループ内の処理を実行してから終了判定を行なっていたため、ループ内の処理が最低一回は実行されます。

しかし、「**While～Wend**」という構文を使うと、ループを実行する前に終了判定を行なうようにして、条件を満たしていなければ処理を一度も実行しないようにすることもできます。

この場合は、

```
While 継続条件
    ループ処理
Wend
```

といった形で記述し、継続条件を満たしていなければ、ループ処理は一度も実行されません。

＊

実際にループ処理を行なう場合にどういった構文を使うかは、「条件の判定をいつ行なうか(処理の前か後か)」「条件判定に使う変数の値がループを抜けたときにどうなるか」といった点を考慮しながら、検討してみてください。

2-7 数当てゲーム

今まで見てきたマクロの基本を確認する意味で、ちょっとしたゲームを作ってみましょう。

今回作るのは、あらかじめ決めておいた適当な数を、ヒントを元に当てる「数当てゲーム」です。

マクロでは、最初に適当な数を決めておき、「入力ダイアログ」で数を入力します。そして、入力された数が決めておいた数より大きいか小さいか、というメッセージを表示して、そのメッセージを元に数を推測していく、という流れにしてみます。

① 正解の数を決める。
②「入力ダイアログ」で数を入力。
③ 正解なら④へ、そうでなければヒントを表示し②へ。
④ 正解メッセージを表示。

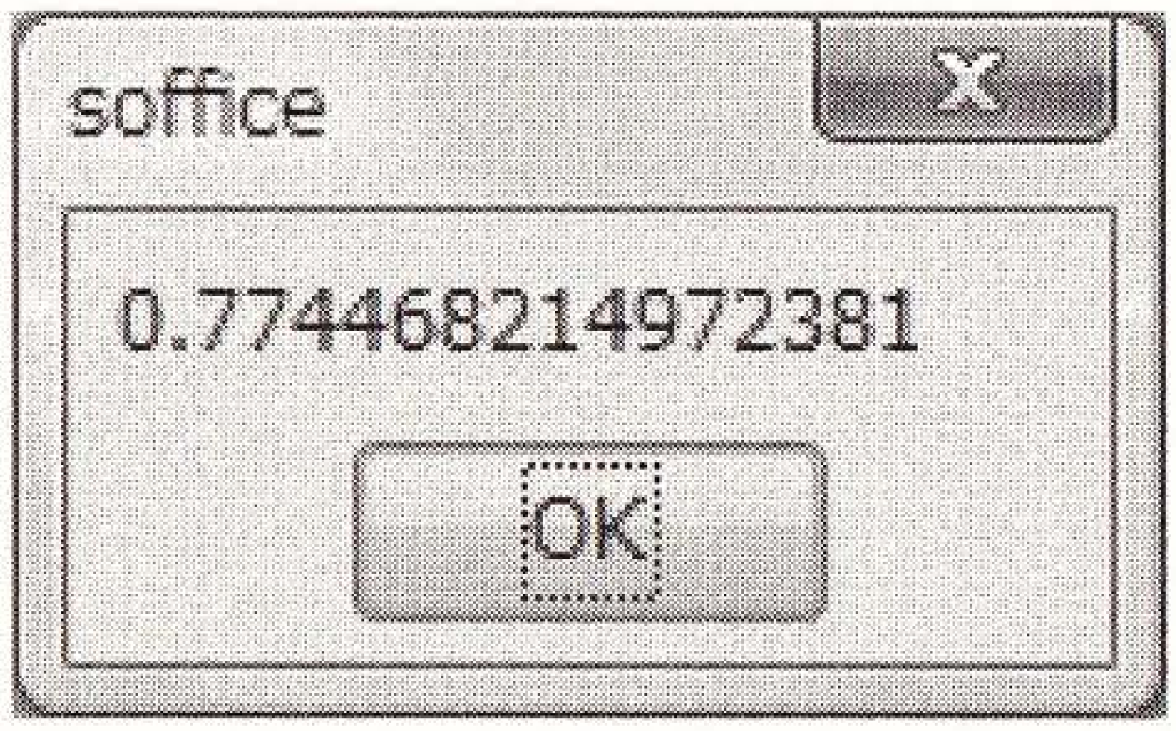
■ 正解の数を設定する

まず、正解の数として「適当な数」を決めるわけですが、これには「**Rnd( )**」という(擬似)乱数を返す関数を使います。

この関数は、「0」以上「1」未満のランダムな数値(実際には、計算で求めたランダムに見える数値)を返すものです。

以下のマクロを実行すると、実行のたびに異なる値が表示されます。

```
Sub Main
    MsgBox(Rnd())
End Sub
```



実行結果

ただ、このままだと適当な「整数」としては使えないので、欲しい数の上限を掛け、小数点以下を切り捨てる関数**「Int( )」**で整数にします。

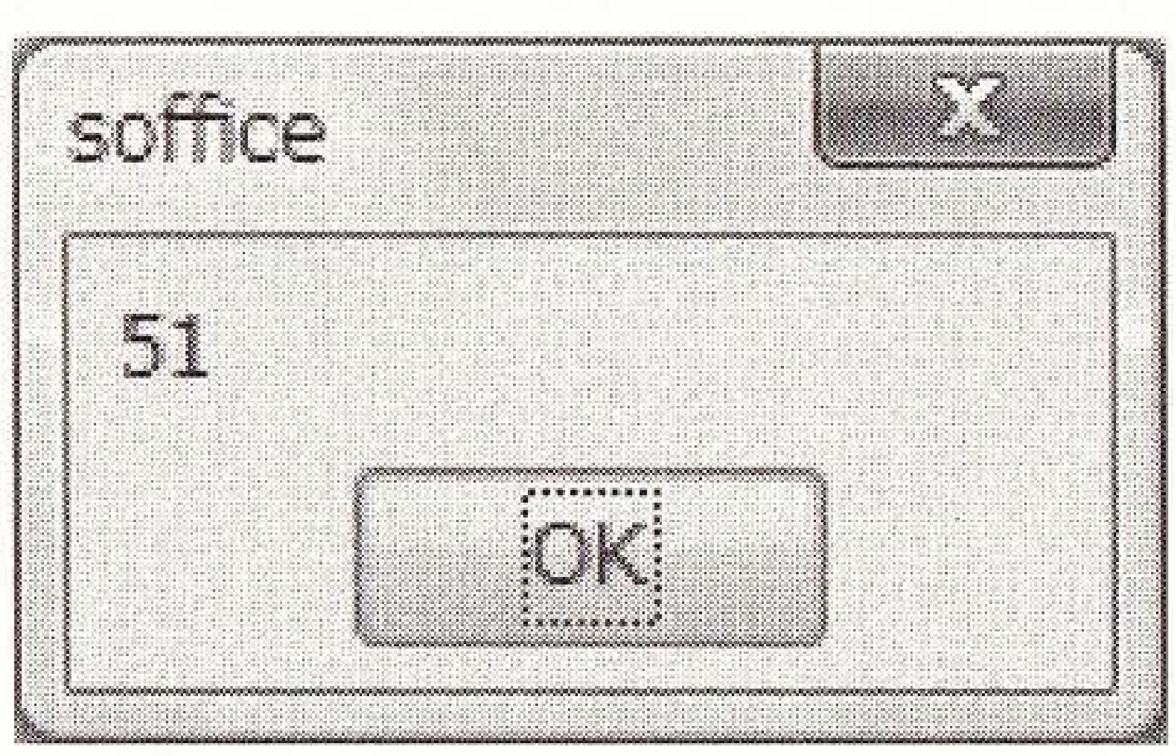
たとえば、「1～100」の間の適当な数が欲しいなら、

```
Int(Rnd() * 100) + 1
```

とするわけです。

先ほどのマクロを以下のように書き換えて何度か実行すると、こんどは(毎回異なる)「1～100」の間の整数が表示されます。

```
Sub Main
    MsgBox(Int(Rnd() * 100) + 1)
End Sub
```



実行結果

今回は、正解の数を「Rnd( )」を使って作りますが、この値は後で何度も使う(入力された値と比較する)ことになるので、変数「num」に入れておきましょう。

■ 判定結果の表示

正解の数を用意できたら、数の入力と判定、判定結果の表示を行なうループに入ります。今回は、「入力した後、正解ならループを抜ける」形で、

```
Do

    数を入力

    入力された数に応じてメッセージ表示

Loop Uintl(入力された数=正解)
```

というループにしましょう。

数の入力は、「InputBox( )」で行ない、結果は変数「inputNum」に格納しておきます。そして、その後、

```
        If (num > inputNum) then
            MsgBox(inputNum & ":正解より小さいです")
        End If

        If (num < inputNum) then
            MsgBox(inputNum & ":正解より大きいです")
        End If

        If (num = inputNum) then
            MsgBox(inputNum & ":正解です")
        End If
```

として、正解の数と比較しメッセージを表示します。

*

以上の処理をまとめ、正解までの入力回数を数える処理も加えると、以下のようになります。

```
Sub Main

  Dim num As Long
  Dim inputNum As Long
  Dim count As Long

  ' 正解の数を決める
  num = Int(Rnd() * 100) + 1

  ' 正解が入力されるまでループ
  Do

        inputNum = InputBox("数を入れてください:")

        If (num > inputNum) then
              MsgBox(inputNum & ":正解より小さいです")
        End If

        If (num < inputNum) then
              MsgBox(inputNum & ":正解より大きいです")
        End If

        If (num = inputNum) then
              MsgBox(inputNum & ":正解です")
        End If

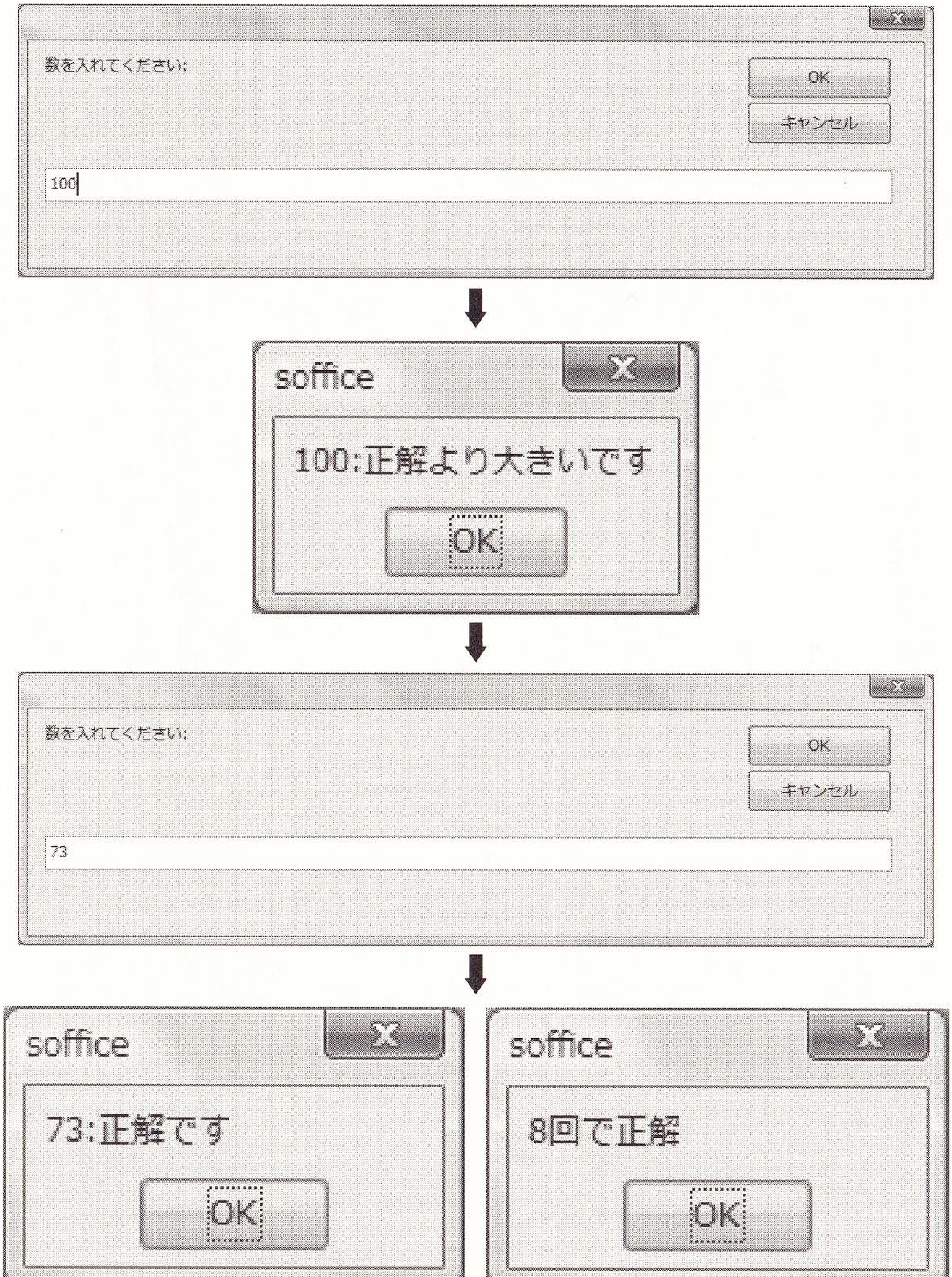
        ' 入力回数カウンタを更新
        count = count + 1

  Loop Until inputNum = num

  ' 正解までの入力回数を表示
  MsgBox(count & "回で正解")

End Sub
```

実行すると、数を入力するダイアログが出てくるので、適当な数を入れて、乱数で決めた数を当ててみてください。



「数当てゲーム」の完成
正解の入力回数を数える処理は、変数「count」で処理

2-8 配列

これまで使ってきた変数では、1つの変数に格納できる値は1つだけでした。

しかし、マクロの作成時には、1つの変数に複数の値を格納できると便利なことがあります。

■ 配列の利点

たとえば、「得点データ」を読み込んで、「平均値」や「偏差値」など、各種統計データを求める処理を考えてみましょう。

「得点データ」は何度も使うので、変数に入れたいですね。しかし、1つの変数に1つの値しか入れられないとなると、

```
Dim score1 As Long, score2 As Long, score3 As Long, score4 As
Long …
```

などと「得点データ」の件数だけ変数が必要になってしまいます。

また、統計処理でも、

```
av = (score1 + score2 + score3 + …) / num
```

などと件数分の変数を並べ計算する、などということはしたくはないでしょう。

こんなときに便利なのが「**配列**」です。

「配列」とは、「複数の値をまとめて格納できる型」で、配列型の変数を1つ用意すれば、その中に得点データを何件でも(もちろん、容量の都合で制限はありますが)まとめて入れることができます。

たとえば、配列型変数「ary」を使うと、

```
ary(1) = 30
ary(2) = 20
ary(3) = 20
```

などという形で、1つの変数「ary」に複数の要素(値)を入れることができます。

変数名の後ろに付ける「()」の中の数字は、**「インデックス」(添え字)**と呼ばれるもので、配列の各要素に付けられる番号です。配列の各要素は、このインデックスで参照します。

```
ary(1) = 30
```

という文は、「配列aryの1番の要素に30を代入する」という意味になります。

■ 配列の宣言

配列の宣言は、変数と同様に、「Dim文」で配列に入れる値の型を指定して行ないます。

配列の宣言時には、インデックスの値の範囲も同時に指定することになっていて、たとえば、

```
Dim ary(1 to 10) As Long
```

という宣言を行なうと、「1から10までの範囲のインデックスをもつ要素数10の整数型配列」、つまり「ary(1)、ary(2)、ary(3)……ary(10)」という形でアクセスできる10個の値を格納するための整数型配列を宣言したことになります。

このインデックスの範囲には任意の数字を指定できるので、たとえば、

```
Dim ary(-10 to 10) As Long
```

とすると、「-10」～「10」のインデックスをもつ配列になります。

ただし、分かりにくくなるので、特に理由がなければ、インデックスは「0」か「1」から始めるのがいいのでしょう。

特に、「1」から始めれば、日常的な意味での「番号」と同じくインデックスの上限の値が配列の要素数になるので、感覚的にも分かりやすくなります。

また、他のプログラミング環境では配列のインデックスが「0」から始まる、と決められている場合も多いので、そうした扱いに慣れている、あるいはプログラムを移植して使うなら「0」からにする、という手もあります。

■ ReDim文

配列のインデックスの範囲は、「Dim文」で宣言した後でも、**「ReDim文」**で配列を宣言し直すことで変更できます。

たとえば、

```
Sub Main

    Dim ary(1 to 5) As Long

    ary(6) = 1

End Sub
```

というマクロを実行すると、「1」から「5」のインデックスをもつ配列として宣言された「ary」に、「ary(6)」として範囲外のインデックス「6」で参照しているため、実行するとエラーになります。

しかし、これを、

```
Sub Main

    Dim ary(1 to 5) As Long

    ReDim ary(1 to 6) As Long

    ary(6) = 1

End Sub
```

とすると、インデックスの範囲を「1」から「6」として「ReDim」されているので、インデックスに「6」を指定しても問題なく実行できます。

■ インデックスの範囲を取得する

また、配列のインデックスの範囲を取得するために、インデックスの下限を返す「**LBound( )**」、上限を返す「**UBound( )**」という関数も用意されていて、これらの関数を使うと配列のインデックスがどの範囲にあるか、調べることができます。

たとえば、

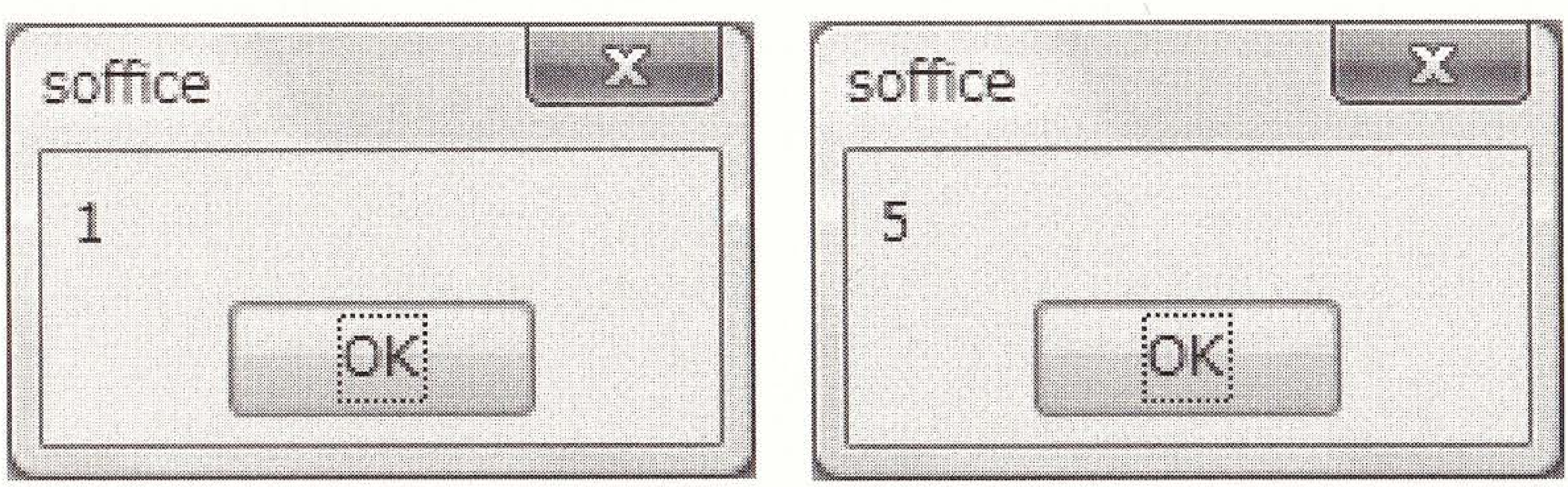
```
Sub Main

    Dim ary(1 to 5) As Long

    MsgBox(LBound(ary))
    MsgBox(UBound(ary))

End Sub
```

というマクロを実行すると、「1」、「5」と配列のインデックスの下限と上限の値が表示されるはずです。



「Lbound( )」「UBound( )」を使えば、インデックスの範囲を調べることが可能

「関数」や「サブルーチン」の引数として配列を渡す場合は、まずこれらの関数で配列のインデックスの範囲を調べてから、配列の各要素を処理することになるでしょう。

2-9 ループ処理②

配列を使う場面では、配列内の全要素を順番に処理する機会が多いと思います。配列に格納されたデータを順番に読み込んで集計したり、逆に順番にデータを入れていったり、という感じですね。

■ For文

そのようなときには、「**For文**」によるループを使うと、スッキリと書くことができます。

これは、「Do文」によるループと同様に、「一定の条件を満たす間」処理を繰り返すループ構文ですが、「最初に変数の値を設定し、処理を行なうたびに変数の値を加算して、変数の値が一定の値になるまで」ループする、という構文です。

```
For 変数 = 初期値 To 終了値
    ループ処理
Next
```

とすると、指定の変数の値を初期値に設定した後に、ループ処理を実行します。

ループ処理を実行し終えたら変数を1ずつ加算し、「For文」に戻ります。そこで変数の値が終了値を超えているか調べ、超えていたらループを終了するわけです。

ただし、「For文」を終えた時点では、変数の値は「終了値＋1」になっている点に注意してください。

たとえば、

```
Sub Main

    Dim ary(1 to 10) As Long
    Dim i As Long

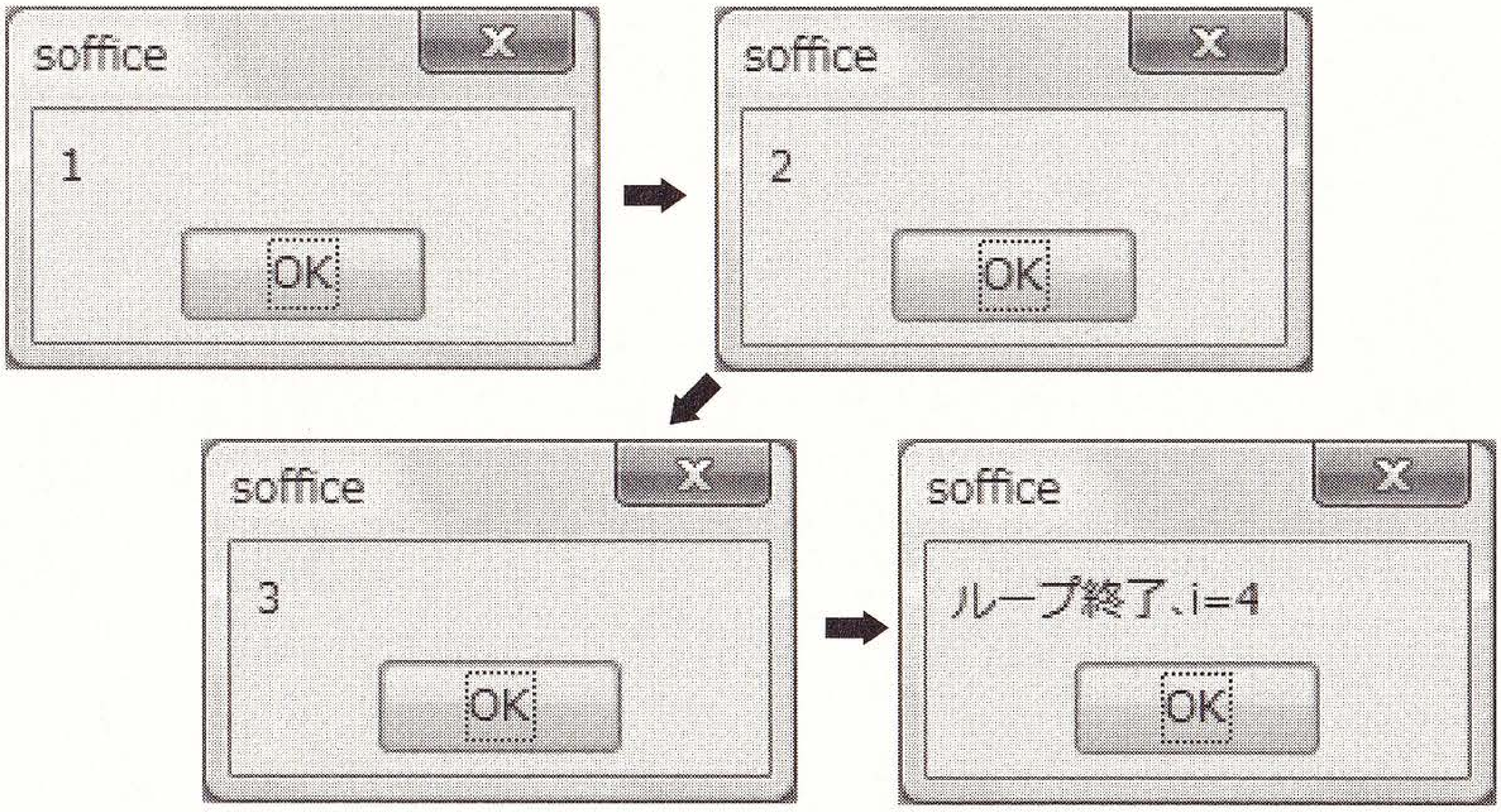
    For i = 1 To 3
```

```
        MsgBox(i)
    Next

    MsgBox("ループ終了、i=" + i)

End Sub
```

というマクロを実行してみましょう。



「For文」を抜けた時点で「1」が加算されるので、最後の「MsgBox( )」では「i=4」となる

ループ内で変数「i」の値を表示しますが、最初に初期値である「1」が表示され、その後「2」、最後に終了値「3」が表示されますね。そして、終了時の「i」の値は「4」になっています。

つまり、ループ内の処理では、「1,2,3という1ずつ増えていく変数iの値」を使って処理をするわけです。これは、まさに「配列内の各要素に対する順次処理」のためにあるような構文ですね。

■ Step増加分

といっても、「For文」は別に配列処理専用の構文というわけではなく、変数の増加分についても任意に指定できます。

変数の増加分を指定するには、「For文」の最後に、**「Step増加分」**を追加します。

たとえば、

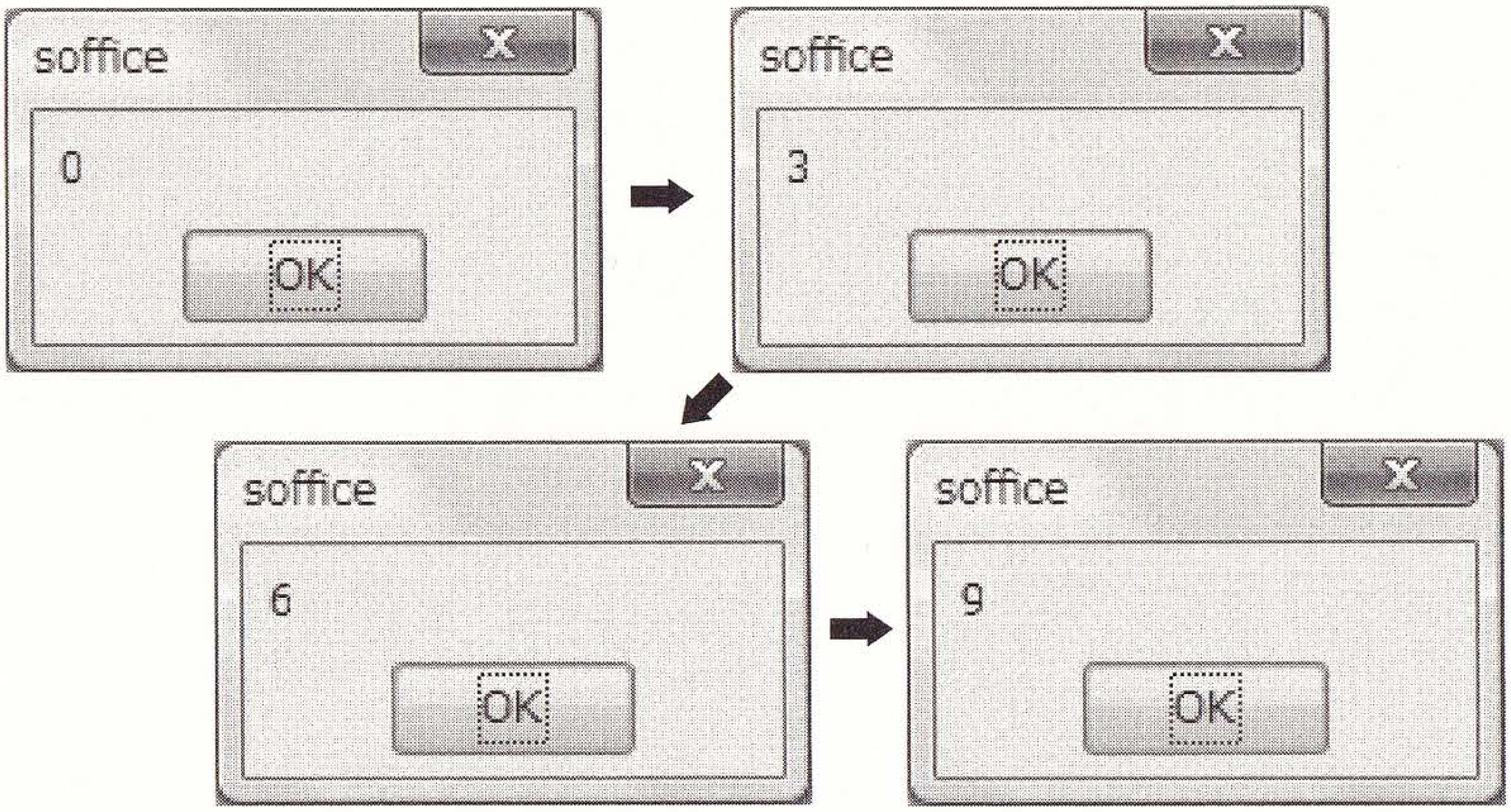
```
Sub Main

    Dim i As Long

    For i = 0 To 9 Step 3
        MsgBox(i)
    Next

End Sub
```

というマクロを実行すると、最初に「0」が、次に3を加えた「3」、さらに「6」「9」と表示され、「i」の値が3ずつ増えていくのが分かると思います。



「0」から「9」まで、「i」の値が3ずつ加算されていく

ちなみに、「Step」で指定する増加分には、負の値を指定することもできます(減少分の指定)。

この場合は、初期値に終了値より大きな値を入れるようにしてください。そうしないと、いつまでも終了値に達せず、ループが終わらないためです。

2-10 多次元配列

配列の宣言では、複数のインデックスをもつ「**多次元配列**」を宣言することができます。

多次元の配列は、イメージとしては配列の各要素が配列になっている「配列の配列」のようなものです。

■ 多次元配列の宣言

この多次元配列の宣言は、

```
Dim ary(1 To 5, 1 To 5) As Long
```

のように、各次元のインデックス範囲を「カンマ」(,)で区切って行ないます。

この例では、各次元に「1」から「5」のインデックスで参照可能な5つの要素がある2次元配列を宣言し、合計で「5×5=25」の要素を格納可能な配列が作られるわけです。

この配列は、各要素が5つの要素をもつ配列になっている配列、または、以下のような5つの要素をもつ配列が、5行積み重なったようなイメージです。

```
ary(1, 1) ary(1, 2) ary(1, 3) ary(1, 4) ary(1, 5)
ary(2, 1) ary(2, 2) ary(2, 3) ary(2, 4) ary(2, 5)
ary(3, 1) ary(3, 2) ary(3, 3) ary(3, 4) ary(3, 5)
ary(4, 1) ary(4, 2) ary(4, 3) ary(4, 4) ary(4, 5)
ary(5, 1) ary(5, 2) ary(5, 3) ary(5, 4) ary(5, 5)
```

「Calc」の表計算シートは、横方向に「A,B,C…」という「列」、縦方向に「1,2,3…」という「行」が並ぶ行列構造になっていて、各セルは「A1セル」などと「列名」と「行名」で指定しますが、2次元配列のイメージはちょうどそれと同じような感じです。

＊

もっとも、通常の配列(一次元)を使った場合でも、「5×5」の2次元構造なら25個の要素をもつ一次元配列「ary」を用意して、

```
Dim ary(1 To 25)

ary(1)からary(5)は(1,1)から(1,5)に対応
ary(6)からary(10)は(2,1)から(2,5)に対応
…
ary(20)からary(25)は(5,1)から(5,5)に対応
```

のような形で表現できるので、それほど多次元配列の必要性はないと言えるかもしれません。

ただ、多次元配列を使うと、2次元や3次元など、多次元構造のデータを分かりやすい形で表現できる場合があります。

たとえば、複数の学生の3教科の得点データを、

```
score(1 To 学生数, 1 To 3)
```

のような2次元配列で扱う手も考えられるでしょう。

こうしておけば、

```
For i = 1 To 学生数

    score(i, 1)の国語の得点を使った処理
    score(i, 2)の数学の得点を使った処理
    score(i, 3)の英語の得点を使った処理

Next
```

のような感じで、簡潔に各学生の得点を処理することができます。

第3章

OpenOfficeマクロに用意されている、便利な機能

「指定したファイルの読み書き」や「ボタンを押すと設定した処理を開始するダイアログ」など、OpenOfficeには、マクロ機能をさらに便利にするさまざまな機能があります。

ここでは、これらの機能について学んでいきましょう。

3-1 CSVの処理

「OOo」では、「通常のテキスト」や「CSV」「TSV」「数字の羅列」など、さまざまな形の文字列を扱います。マクロにも文字列を扱うための関数が各種用意されているので、試してみましょう。

■文字列を分割して、配列として返す

まずは、「CSV」を扱うときに便利な「**Split( )**」関数です。これは、文字列を特定の「区切り文字列」で分割し、その配列を返すものです。

たとえば、

```
item1,item2,item3
```

という文字列を、「,」を区切り文字列として分割し、各項目を「配列」として返すなら、

```
Split("item1,item2,item3", ",")
```

とします。

すると、

```
item1
item2
item3
```

という文字列を要素とする配列が返されるわけです。

2番目の引数に指定する「区切り文字列」には、任意の文字列が指定できるので、

```
item1abcitem2abcitem3
```

といった文字列を「abc」を「区切り文字列」に分割する、といったこともできます。

```
Sub Main

    Dim items() As String

    items = Split("item1abcitem2abcitem3", "abc")

    MsgBox items(1)

End Sub
```



文字列が「abc」を区切りとして分割される

■配列の文字列をCSV文字列にまとめる

こんどは逆に、「配列の文字列」を1つの「CSV文字列」にまとめてみましょう。

「**Join( )**」を使うと、「配列の文字列」を指定の「区切り文字列」を挟みながら1つの文字列にまとめることができます。

たとえば、以下のマクロを実行すると、配列の各要素の文字列が「,」を挟んだ「CSV」として連結されて表示されます。

```
Sub Main

    Dim items(0 To 2) As String
    Dim str As String

    items(0) = "item1"
    items(1) = "item2"
```

```
    items(2) = "item3"

    str = Join(items, ",")

    MsgBox str

End Sub
```



配列が、「,」を“区切り文字”として連結される

「Join( )」の2番目の引数に「" "」を渡すと、「区切り文字なし」で、配列の各要素を単純に連結した文字列が得られます。

「CSV」の処理は行なう機会も多いと思うので、これまで試してみたマクロをいろいろ修正しながら実行し、「Split( )」と「Join( )」の扱いに慣れておきましょう。

■文字列の前後の空白を取り除く

「CSV」から切り出した文字列には、先頭や末尾に「空白」が付いてくることがあるかもしれません。そんな先頭や末尾の空白を取り除くのが、「**Trim( )**」です。

この関数は、渡された文字列の先頭と末尾にある空白を除去した文字列を返します。

以下のマクロを実行すると、文字列「str」の先頭と末尾の空白が除去されて、「:te st:」と表示されます(中間の“空白”が取り除かれていない点に注意)。

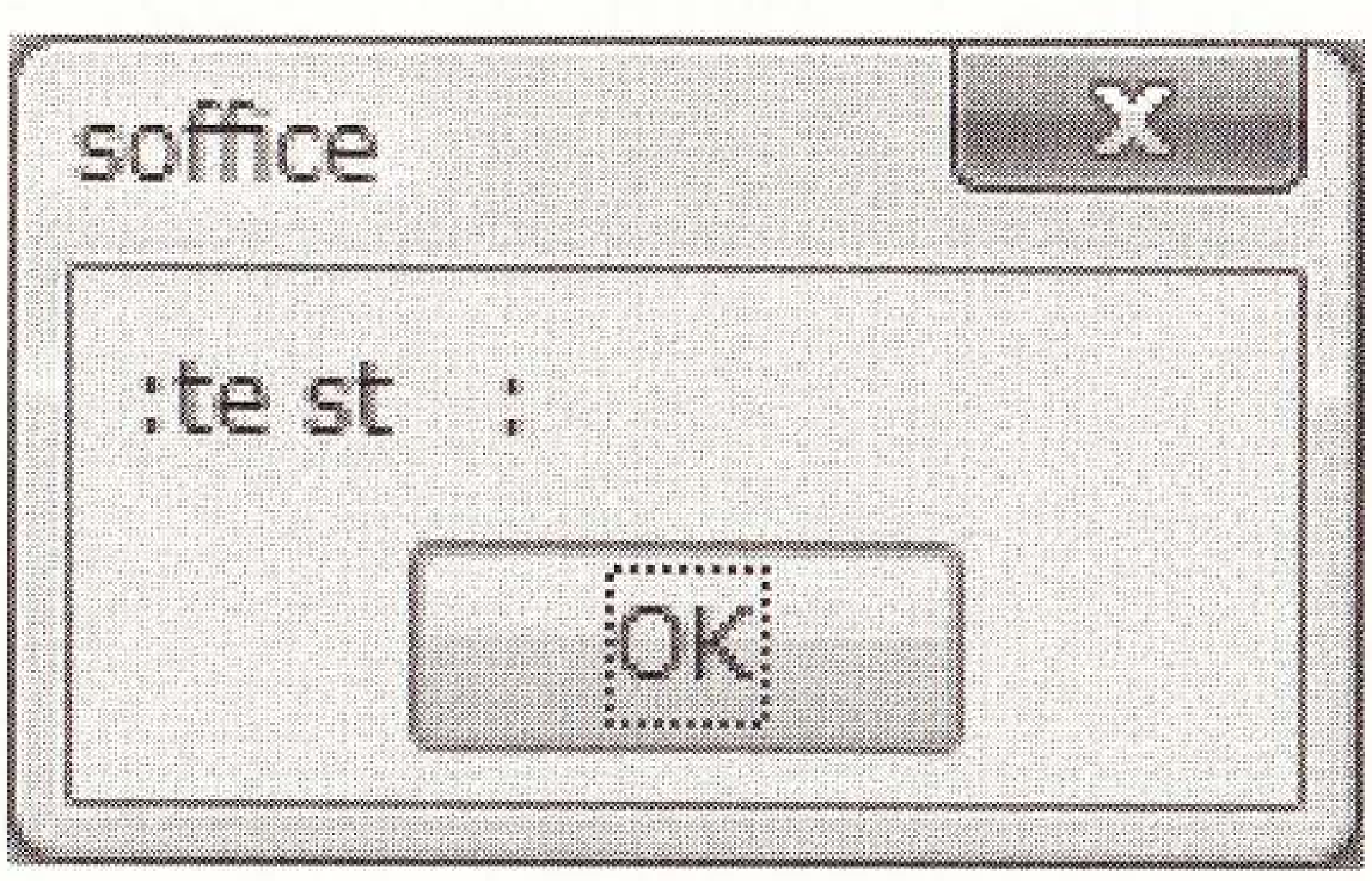
```
Sub Main
    Dim str As String
    str = "   te st   "
    MsgBox ":" & Trim(str) & ":"
End Sub
```



文字列の「文頭」と「文末」の空白が削除される

また、「Trim( )」は「先頭」「末尾」両方の「空白」を取り除きましたが、**「LTrim( )」**を使うと、先頭にある「空白」のみ取り除き、**「RTrim( )」**を使うと末尾の「空白」のみを取り除くことができます。

先のマクロの「Trim(str)」の部分を、「LTrim(str)」や「RTrim(str)」に変更して試してみてください。



「LTrim(str)」の場合



「RTrim(str)」の場合

3-2 ファイルの読み書き

マクロには、「ファイルの読み書き」に関する機能も用意されています。

ファイル読み書き機能を使うと、コンピュータに保存されているファイルやネット上のファイルにアクセスして内容を取得したり、情報を保存したりできるようになるので、さまざまなデータを「OOo」に取り込むことが可能になります。

ファイルの読み書きには、**「ファイル番号」**という番号を使います。

これは、「OOo」で管理しているファイル・アクセスの番号で、**「Freefile( )」**関数で使用可能な番号を取得できます。

ファイル番号を取得したら、①**「ファイル・パス」**と、②読み込みや書き込みといった**「モード」**、そして③取得した**「ファイル番号」**を指定して、**「Open文」**でファイルを開きます。

たとえば、以下のようにすると、「d:¥test.txt」というファイルを読み込み用に開くことができます。

```
Dim iFile As Integer

iFile = Freefile

Open "d:¥test.txt" For InPut As #iFile
```

ファイルを開いたら、続いて「Line Input」などファイル・アクセス用の構文に「ファイル番号」を渡してファイルを読み取り、最後に「Close」でファイルを閉じます。

■テキスト・ファイルの行データを調べる

試しに、適当なテキスト・ファイル(あまり大きくないもの)を用意して、適当な場所に配置してください。

その後、以下のマクロの「d:¥test.txt」の部分をテキスト・ファイルのパスに書き換えて実行してみましょう。

```
Sub Main

    Dim iFile As Integer
    Dim strLine As String

    iFile = Freefile

    Open "d:¥test.txt" For InPut As #iFile

    Line Input #iFile, strLine

    MsgBox strLine

    Close #iFile

End Sub
```

実行すると、最初の一行(最初の改行までの部分)が表示されたと思います。



テキスト・ファイルの一行目の内容が表示される

「Line Input文」は、指定のファイルから一行読み込んで、現在の位置を次の行の先頭に進めます。ただし、すでにファイルの終わりに達しているのに「Line Input」を呼び出すとエラーになるので、注意してください。

■ファイルのデータが残っているかを調べる

「Line Input文」でファイルを読み込むときは、前もってファイルの終わりに達しているか調べる**「eof文」**で、ファイルにまだデータが残っているか、確認するようにしましょう。

「eof」は、ファイルの終わりに達すると「True」を返すので、テキスト・ファイル全体を1行ずつ読み込んで順次処理するには、以下のようにします。

数行の適当なテキスト・ファイルを作って保存したら、そのファイルの場所を「d:¥test.txt」の位置に指定して実行してみてください。

```
Sub Main

    Dim iFile As Integer
    Dim strLine As String

    iFile = Freefile

    Open "d:¥test.txt" For InPut As #iFile

    While not eof(iFile)

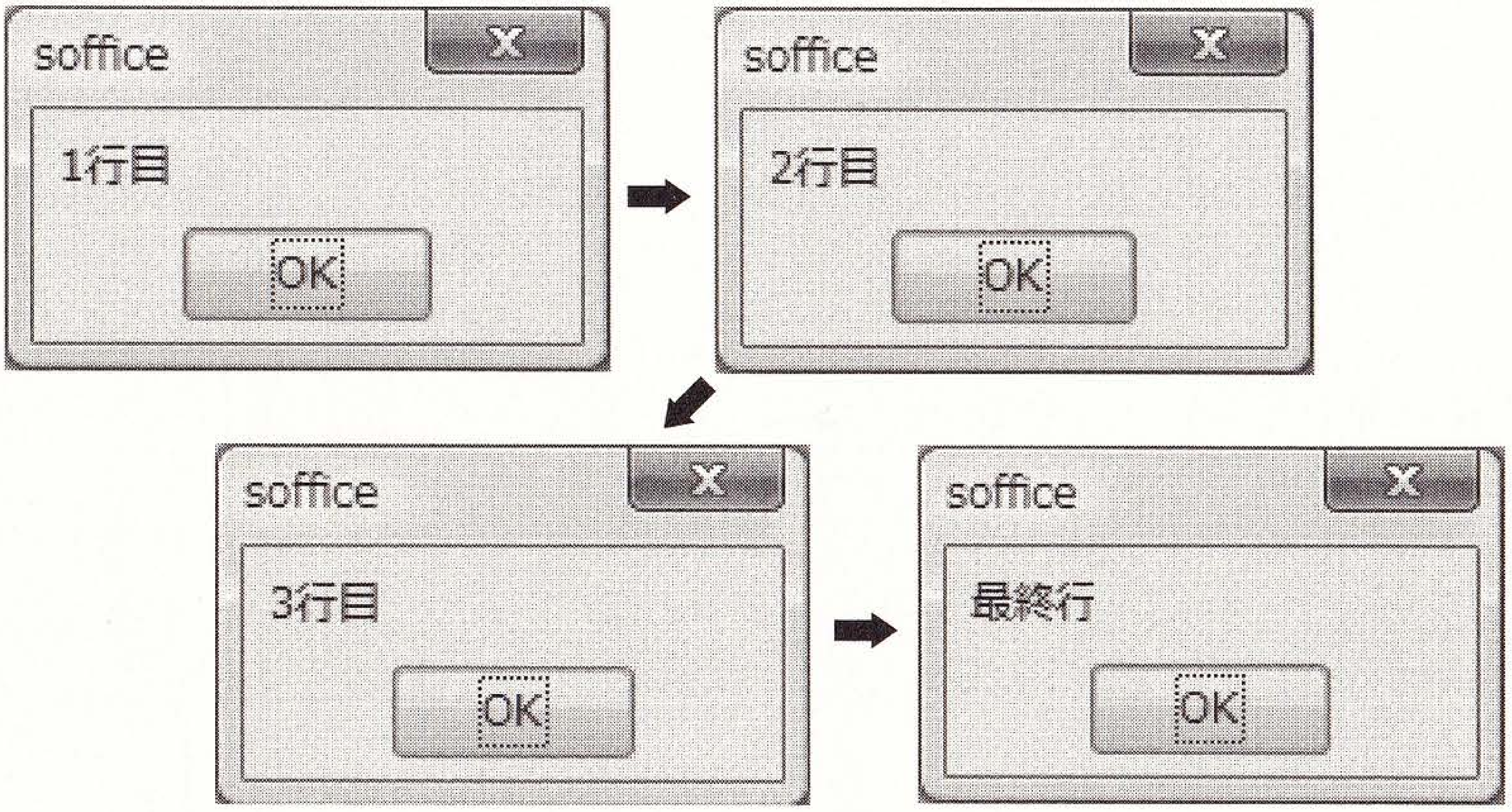
        Line Input #iFile, strLine

        MsgBox strLine

    Wend

    Close #iFile

End Sub
```



1行ずつデータが読み込まれる

3-3 オブジェクト

「OOo」に限らず、最近のソフトでは多数の「**オブジェクト**」が活躍します。

何か行ないたい処理がある場合は、ソフトやシステムに対してオブジェクトを要求し、そのオブジェクトが公開している機能で処理を実現する、という形です。

「オブジェクト」とは、いくつかの「**プロパティ**」(外部から参照できる属性。オブジェクトの状態や設定を取得、変更する)や「**メソッド**」(外部から利用できる機能。サブルーチンや関数として公開され、呼び出すことでオブジェクトに一定の動作を行なわせる)をひとまとめにしたものです。

■オブジェクトを利用する流れ

たとえば、多くのシステムでは動画や音楽を再生する「メディアプレイヤー」もオブジェクトとして公開され、各種のプログラミング環境から利用できるようになっています。

この「メディアプレイヤー・オブジェクト」をプログラムで使う場合は、

[1]システムに対して「メディアプレイヤー・オブジェクト」を要求。
[2]取得したメディアプレイヤーのプロパティを変更し、「表示形式」や「大きさ」などを設定。
[3]「読み込みメソッド」で再生するデータを読み込む。
[4]「再生メソッド」で再生開始。

といった流れで再生を行なうことになるでしょう。

ただ、この場合も、システムが返すのは、「オブジェクト」そのものとは限りません。場合によっては、オブジェクトの一部分にアクセスするための**「インターフェイス」**を返す場合もあります。

■オブジェクトとインターフェイスの違い

「インターフェイス」は、「オブジェクトが備える機能や設定」を定義して操作するためのリモコンのようなものです。

「メディアプレイヤー・オブジェクト」を操作するためのインターフェイスであれば、「表示形式」や「音量」などを設定する“プロパティ”や、「再生」「停止」といった“メソッド”をもつことになるでしょう。

インターフェイスを使うぶんには、基本的にオブジェクトと同じ感覚で使えるので、返されるのが「オブジェクトそのもの」か「オブジェクトを扱うためのインターフェイス」か、といった違いはあまり意識する必要はありません。

ただ、システムの側にとっては、インターフェイスを返す形にすると、「オブジェクトそのもの」を返す必要がなくなる(オブジェクトを直接操作されずにすむ)ので、オブジェクトが提供する機能や属性をどういった形で実現するか、自由に決めやすくなります。

オブジェクトの機能の詳細を変更したり、複数のオブジェクトを用意して状況に応じて適した「オブジェクトを操作するためのインターフェイス」を渡す、といったことが行ないやすくなるわけです。

「プレイヤーを使わせてくれ」と言われても、プレイヤーそのものを渡してしまうのではなく、「機能を限定したリモコン」(プレイヤーを使うためのインターフェイス)を渡して、「プレイヤーは企業秘密で貸せないから、このリモコンを通して使ってくれ」というような感じでしょうか。

3-4 インターフェイス

「OOo」が公開している**「インターフェイス」**をマクロで実際に使ってみましょう。

■サービス

「OOo」のマクロでは、「OOo」が提供するインターフェイスを**「サービス」**という形で取得します。

サービスはインターフェイスと同様に、いくつかの機能を提供しますが、「ひとつのサービス」が「複数のインターフェイス」をもっていることもあります。また、サービスの中にサービスが含まれていることもあります。

とはいえ、マクロから使う場合は、サービスの中に含まれるインターフェイスについては特に意識する必要はなく、サービスを取得したら、直接サービスに含まれるインターフェイスのメソッドを呼び出すことが可能です。

サービスを作る**「CreateUnoService( )」**関数に、サービスの名前を渡すと、そのサービスが作られて返されるので、そのサービスがもっているインターフェイスを通して、「OOo」の機能を利用するのです。

■フォルダ選択ダイアログの表示

サービスの中に、「ディレクトリ選択ダイアログ」を提供する**「com.sun.star.ui.dialogs.FolderPicker」**というサービスがあるので、まずはこれを試してみましょう。

サービスの利用は、

[1]「Object型」変数を用意する。
[2]「CreateUnoService( )」で作ったサービスを、「Object型」の変数に格納する。
[3]サービスのもつインターフェイスを通してサービス機能を利用する。

という流れで行ないます。

ここで試してみる「FolderPicker」サービスには、「**XFolderPicker**」というインターフェイスがあり、この「XFolderPicker」インターフェイスの各メソッドで、ダイアログを表示したり、選択されたダイアログを取得したりします。

試しに、以下のマクロを実行してみてください。

```
Sub Main

    Dim fp As Object

    fp = createUnoService("com.sun.star.ui.dialogs.FolderPicker")

    fp.execute()

    MsgBox(fp.getDirectory())

End Sub
```

実行すると「フォルダ選択ダイアログ」が出てきて、フォルダを選んでダイアログを閉じると選択されたフォルダが表示されたと思います。



「フォルダ選択ダイアログ」が表示される

■「サービスの一覧」「インターフェイスの一覧」を取得する

マクロを開発する際は、取得したサービスが実際にどんなサービスやインターフェイスをもっているか、調べたくなることもあるでしょう。

サービスは、内部に複数のサービスやインターフェイスを含むことができるので、調べてみると意外な機能をもっていることもあります。

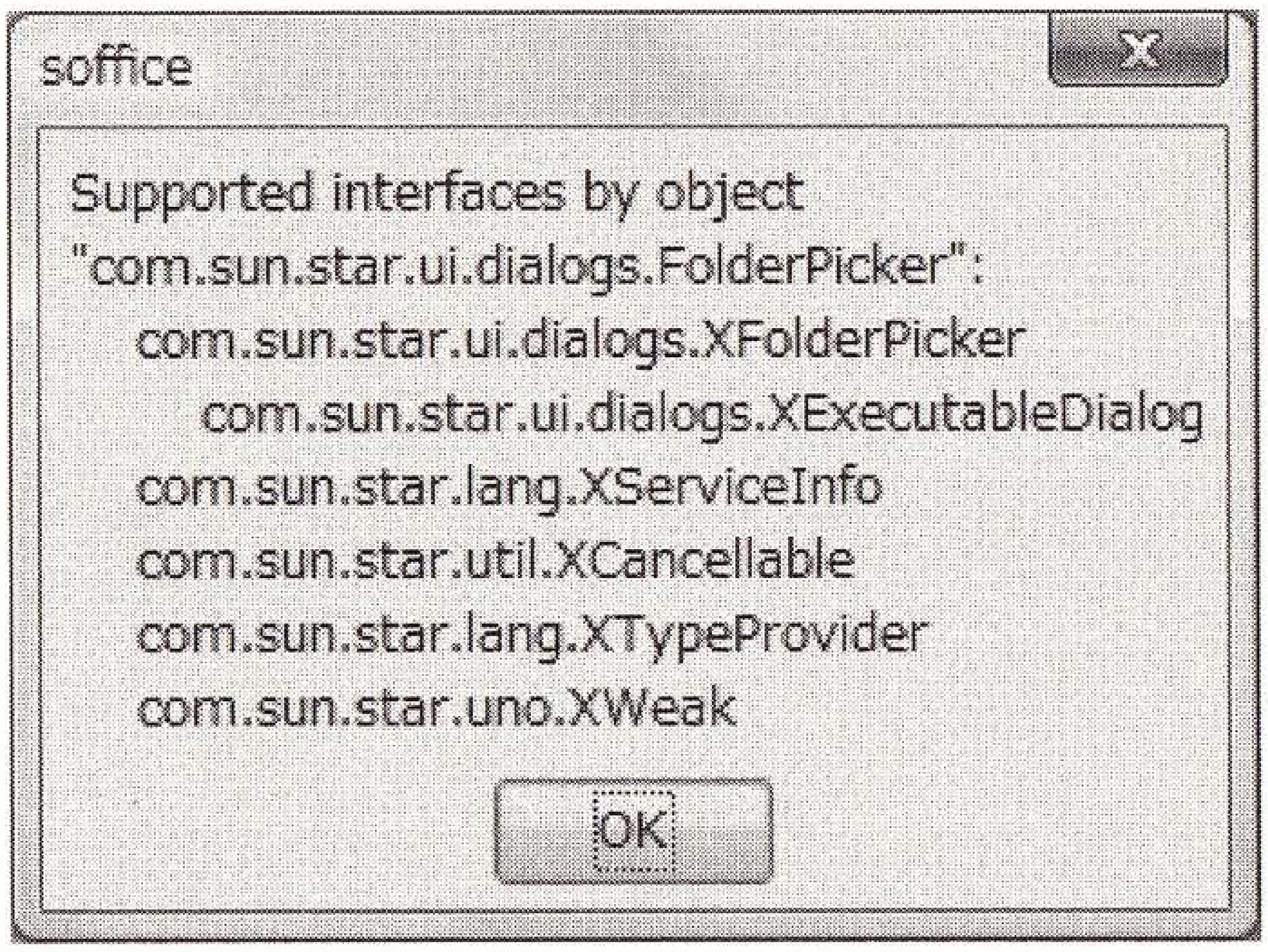
すべてのサービスが実装すべきとされている「**XServiceInfo**」インターフェイスの「**getSupportedServiceNames( )**」メソッドを使うと、使用可能な「サービスの一覧」を取得できます。

さらに、「**Dbg_SupportedInterfaces**」では「インターフェイスの一覧」を文字列として取得できます。

取得してみたものの、使い方がよく分からないサービスは、まず「Dbg_SupportedInterfaces」でインターフェイスの一覧を取得し、各インターフェイスについて調べてみるのも手でしょう。

例として、今回使った「FolderPicker」がもっているインターフェイスを見るには、以下のようにします。

```
Sub Main
    Dim fp As Object
    fp = createUnoService("com.sun.star.ui.dialogs.FolderPicker")
    MsgBox(fp.Dbg_SupportedInterfaces)
End Sub
```



インターフェイスの一覧が表示される

また、同様に「**Dbg_Properties**」でサービスがもっているプロパティを、「**Dbg_Methods**」でメソッドの一覧を取得できます。

ただ、プロパティやメソッドは数が多いですし、名前だけ列挙されても分かりにくいので、個々のプロパティやメソッドについては、インターフェイスやサービスの一覧を取得した後、リファレンスで個別に確認したほうがいいかもしれません。

3-5 ダイアログの作成

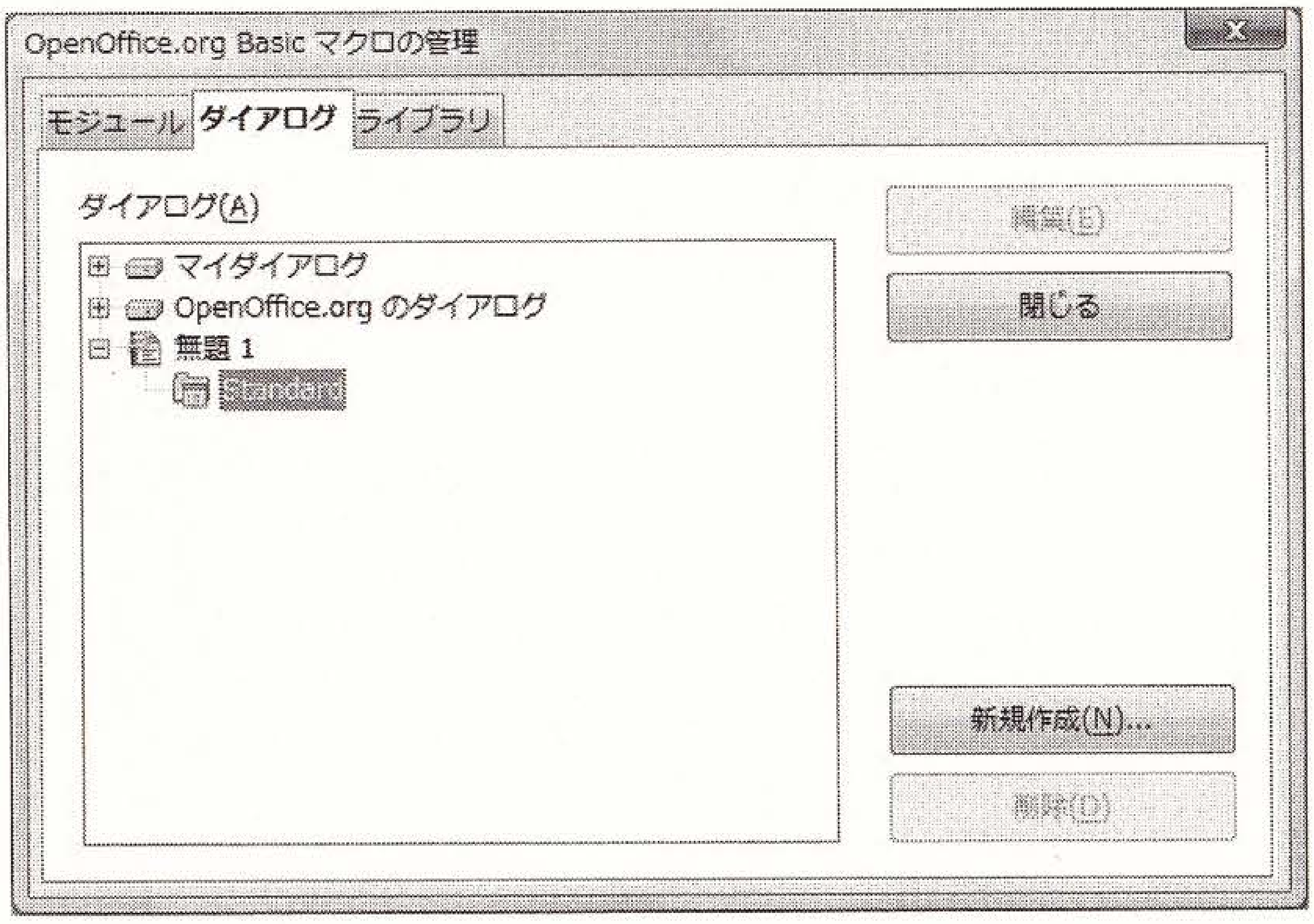
「フォルダ選択ダイアログ」に続いて、こんどは独自のダイアログを作ってみましょう。

「OOo」には、「ボタン」や「ラベル」などのGUI部品を自由に配置して、独自のダイアログを作る機能が用意されています。このダイアログの機能を使うと、ちょっとした「設定フォーム」などを簡単に作って、マクロからも利用できるので便利です。

■ダイアログの作成

ダイアログを作るには、**「ダイアログ・エディタ」**を使います。

「Calc」などのアプリケーションを起動したら、ツールの「マクロ」メニューから「ダイアログの管理」を選んでください。



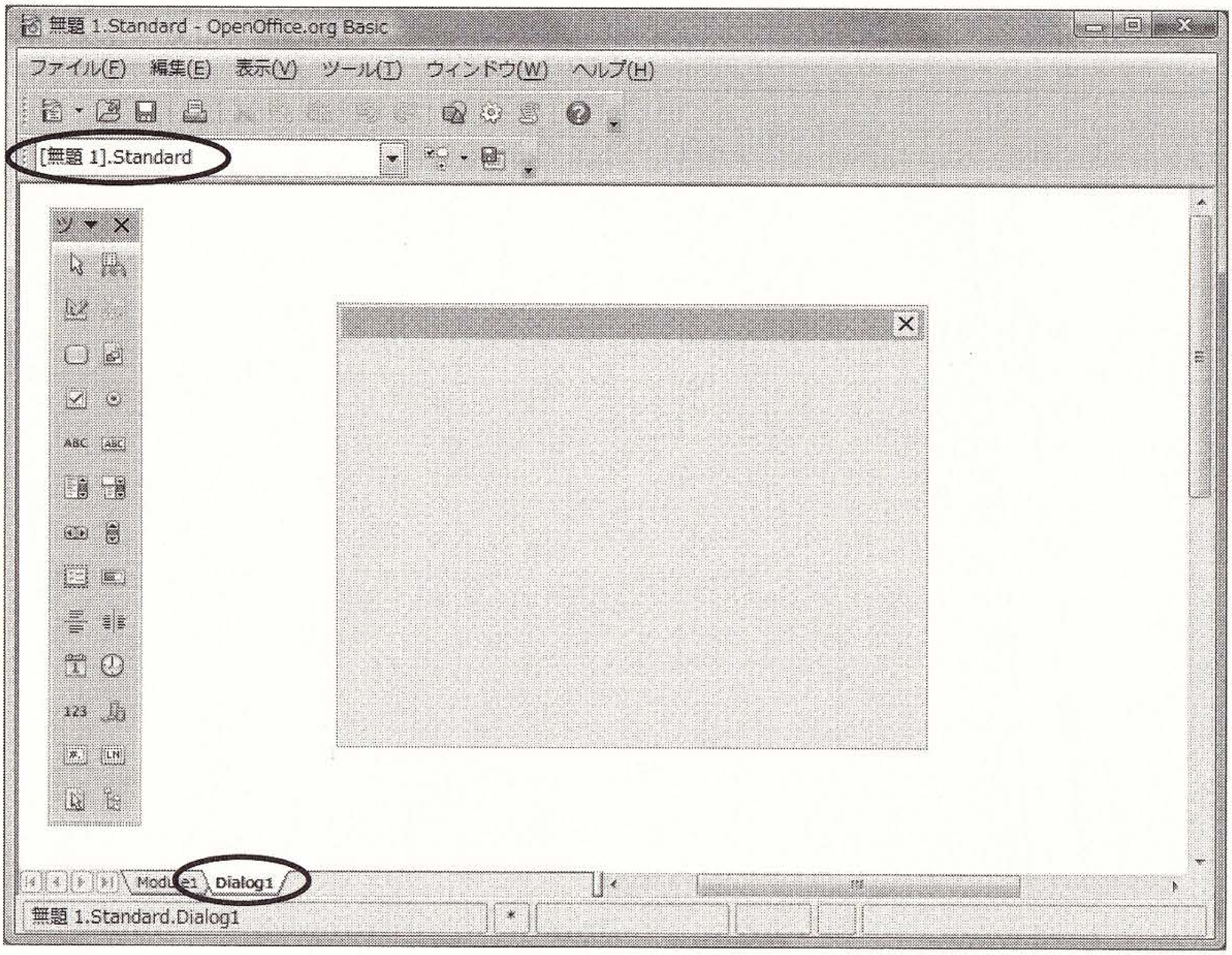
「ダイアログの管理」画面

「ダイアログの管理」画面では、「無題1」など、現在の文書のアイコンをクリックし、「新規作成」します。

ダイアログの名前の指定は、とりあえずデフォルトの「Dialog1」のままにしておきましょう。

「OK」ボタンをクリックすると、文書の下に「Standard」フォルダが作られ、その中にダイアログが作られます。

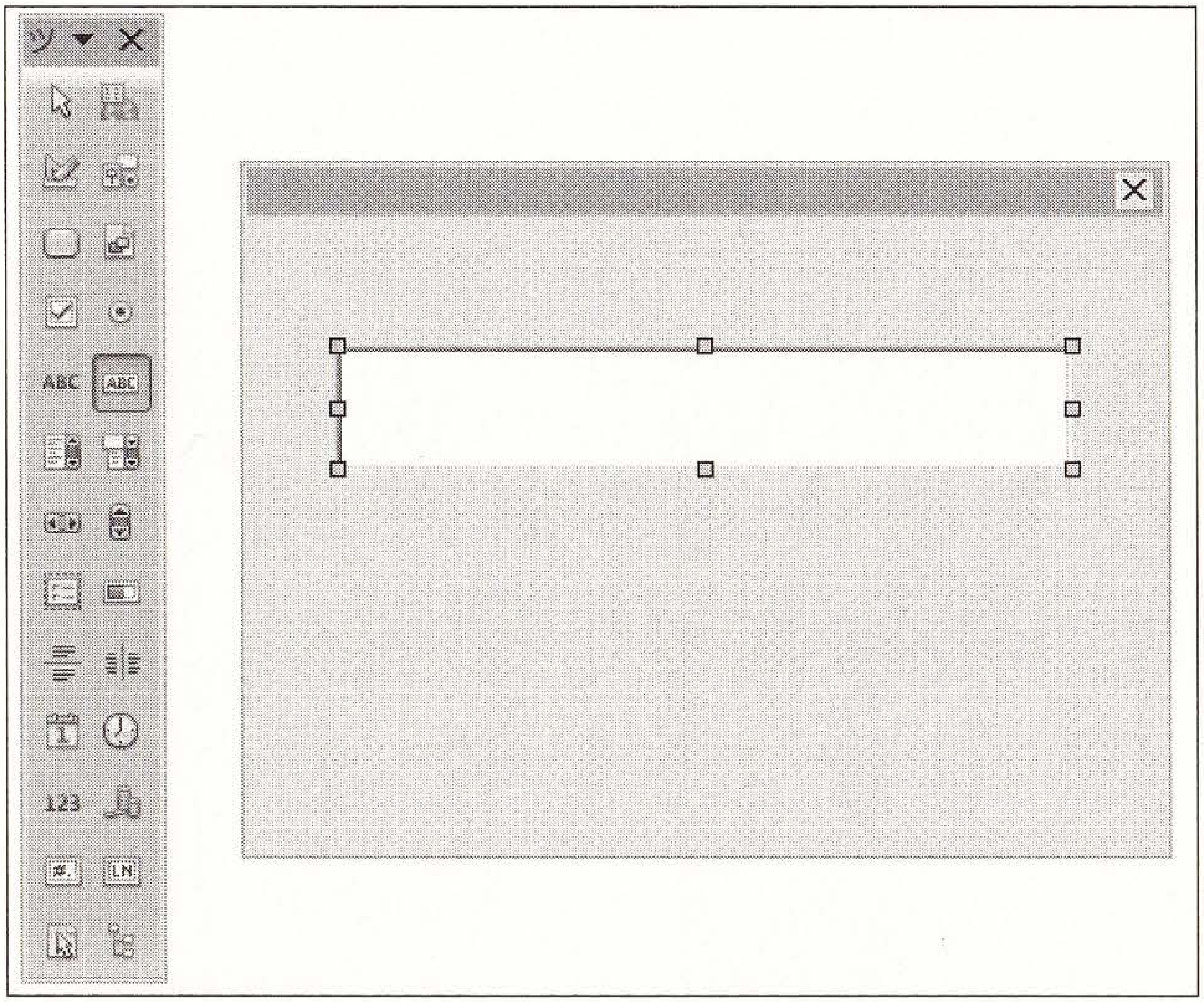
作られたダイアログが選択状態になっていることを確認し、「編集」ボタンでダイアログにGUI部品を配置する「エディタ」画面に入ってください。



「エディタ」画面

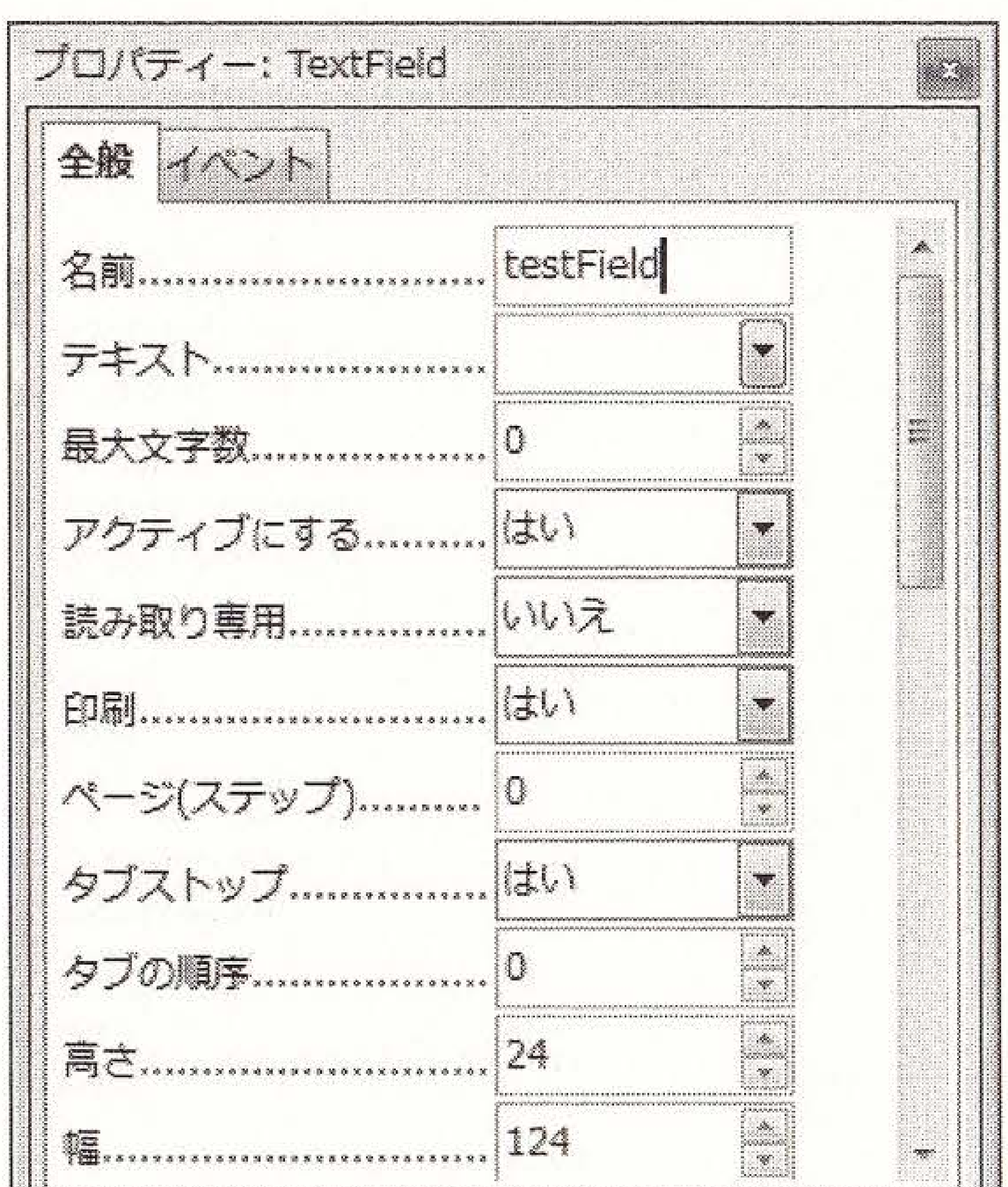
「エディタ」画面では、「ツール・ボックス」に、ダイアログに配置できるGUI部品が並んでいます。

とりあえず、「テキスト・ボックス」(入力欄)をクリックして選択したら、ダイアログの上で適当にドラッグして、配置してください。



「テキスト・ボックス」を配置

「テキスト・ボックス」を配置したら、左クリックで「プロパティ」設定画面を呼び出し、名前を「testField」など適当な名前に変更しておきます。



「プロパティ」設定画面

以上でテスト用のダイアログが出来たので、マクロから表示してみましょう。「マクロの編集」画面に入ったら、以下のマクロを実行してみてください。

```
Sub Main

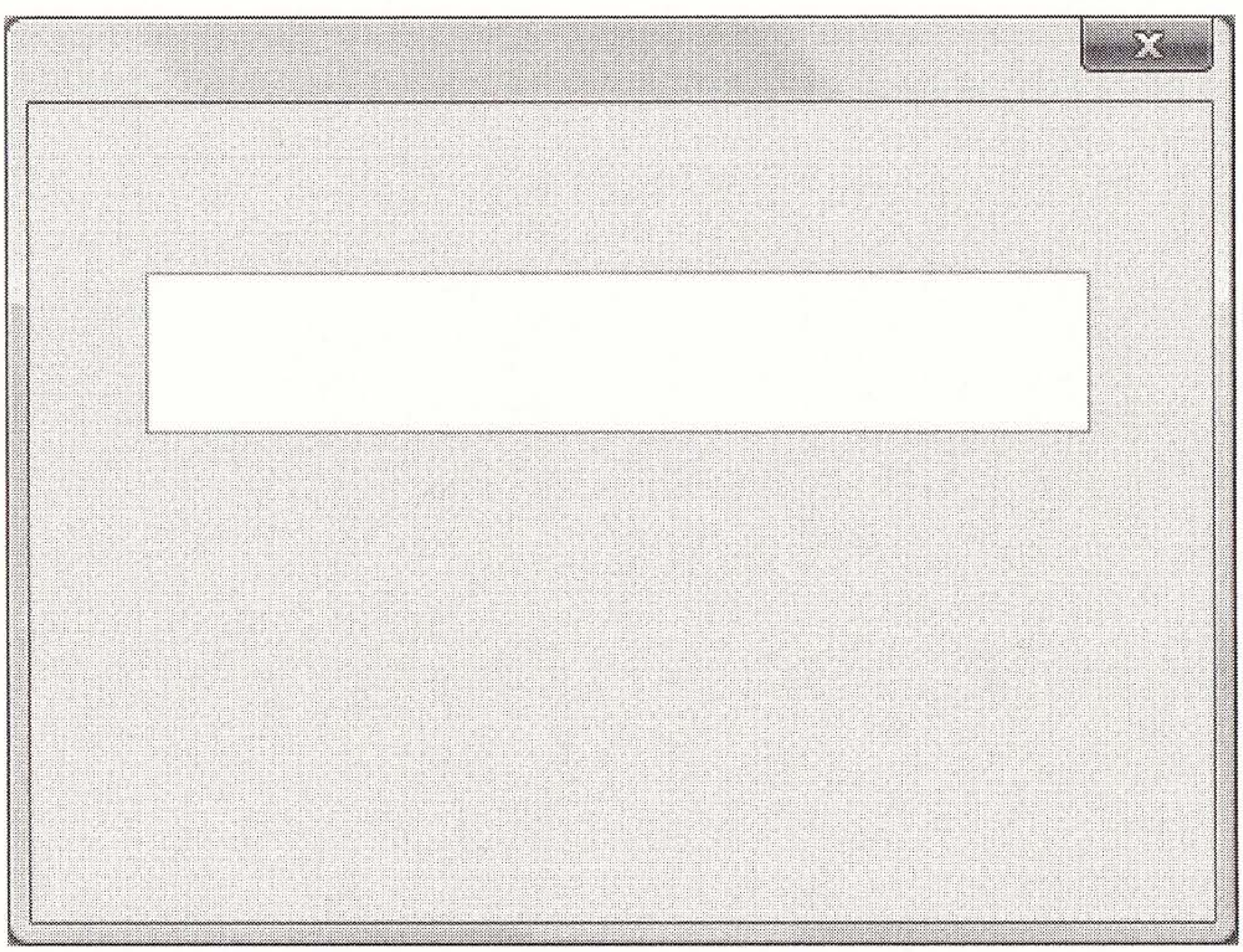
    Dim dlg As Object

    DialogLibraries.LoadLibrary("Standard")

    dlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1)
    dlg.Execute()

End Sub
```

実行すると、先ほど作ったダイアログが表示されたと思います。



作ったダイアログが表示される

このマクロでは、まずダイアログを含む「Standard」ライブラリを読み込み、「**CreateUnoDialog( )**」でダイアログの名前を指定してダイアログを作り、最後に「**Execute( )**」メソッドでダイアログを表示しています。

「Execute( )」はダイアログを表示するメソッドですが、ダイアログを表示すると、何らかの方法でダイアログが閉じられるまでマクロに制御を返さない(ダイアログ表示中はマクロの処理が止まる)ので、注意が必要です。

「Execute( )」はダイアログが閉じられると終了し、返り値でダイアログが閉じられた原因を知ることができます。

もしエラーになる場合は、「ダイアログの管理」メニューで再度「ダイアログの管理」画面を呼び出して、ダイアログの名前を確認してみてください。

■ダイアログのGUI部品にアクセスする

次に、ダイアログに配置したGUI部品、先ほど作ったダイアログのテキスト・ボックス「testField」にアクセスしてみましょう。

ダイアログ上のGUI部品にアクセスするには、GUI部品の名前を引数に「**GetControl( )**」メソッドを呼び出します。

以下のマクロでは、先ほどのダイアログを表示した後、ダイアログに配置したテキスト・フィールドを取得し、その内容(Text)を表示します。

実行したら、ダイアログ内のテキスト・フィールドに適当な文字列を入れて、ダイアログを閉じてみてください。

```
Sub Main

    Dim dlg As Object
    Dim fld As Object

    DialogLibraries.LoadLibrary("Standard")

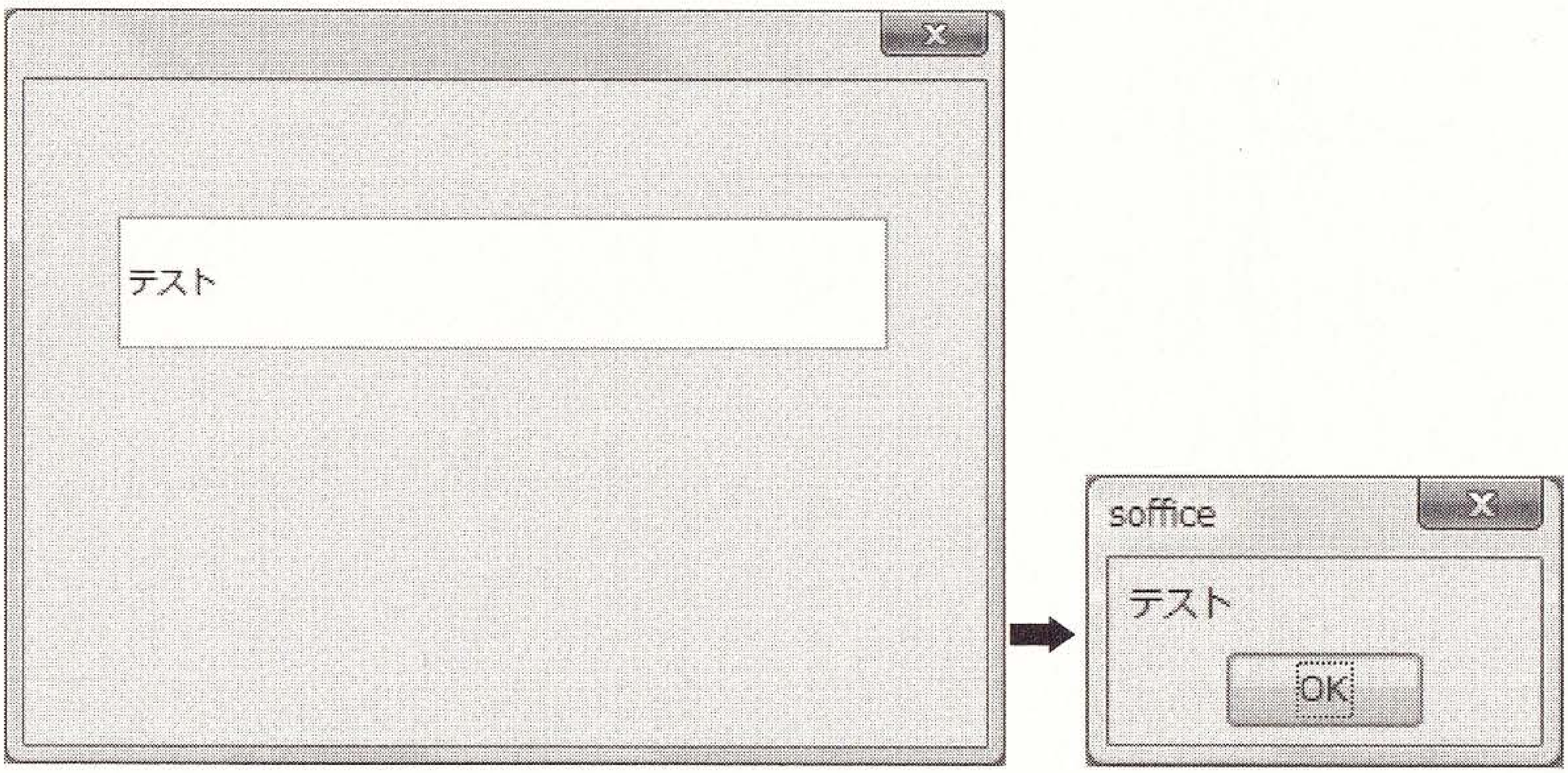
    dlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1)
```

```
    dlg.Execute()

    fld = dlg.GetControl("testField")

    MsgBox fld.Text

End Sub
```



テキスト・フィールドに入力した内容が、メッセージ・ボックスに表示される

■ボタンの追加

最後に、ダイアログに「OK」ボタンを追加してみましょう。

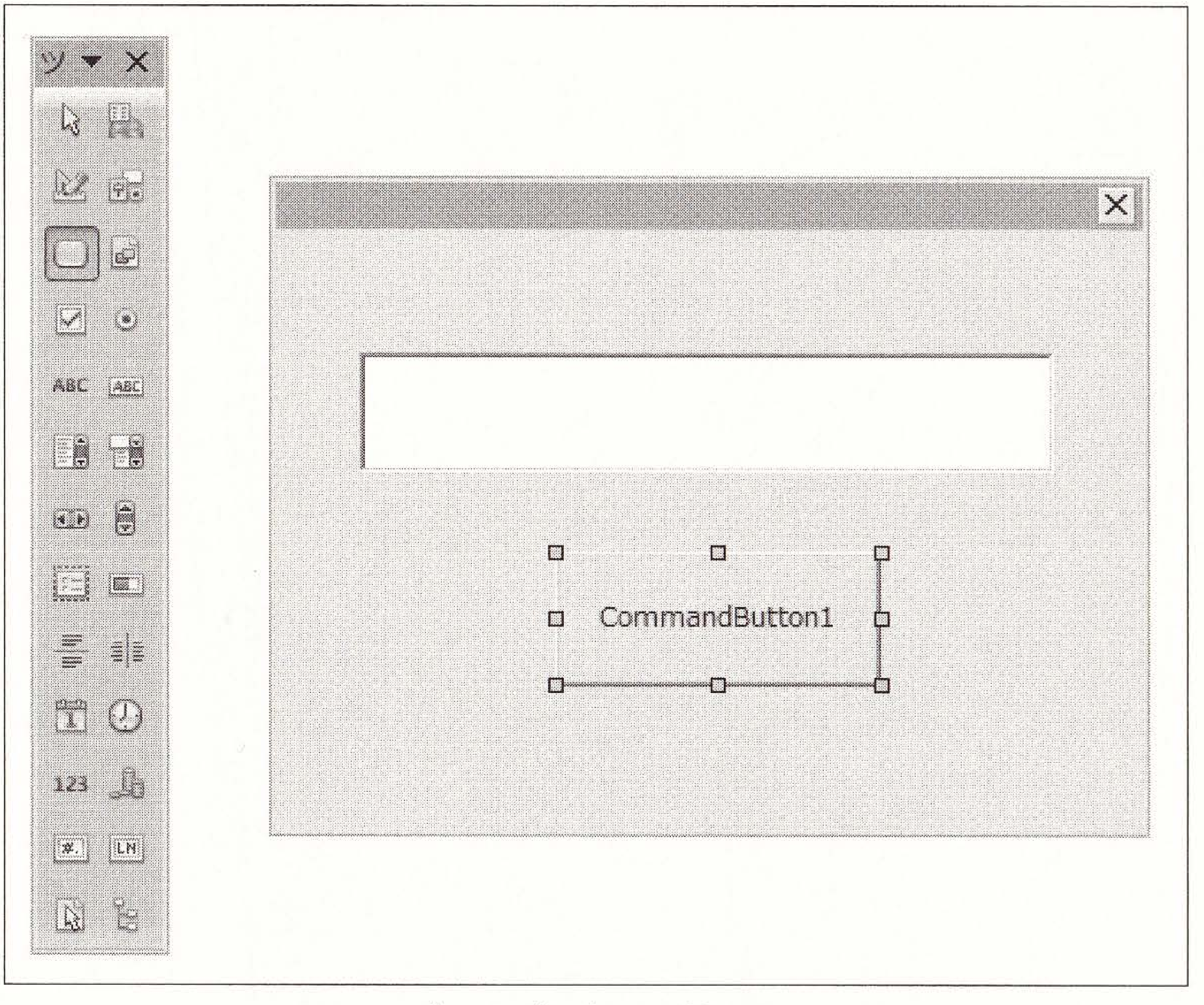
ダイアログに「OK」ボタンを追加しておくと、そのボタンをクリックしたときにダイアログを閉じて、「Execute( )」の返り値で、「OK」ボタンからダイアログが閉じられたことを確認できるようになります。

つまり、「OK」ボタンを追加したダイアログでは「Execute( )」で表示した結果、返される値を見ることで、「OKボタンをクリックして終了した」のか「それ以外の方法で終了した」のか、見分けることができるようになるわけです。

「OK」ボタンで終了したならダイアログで指定された処理を実行し、そうでなければキャンセルされたものとして何もしない。このような**「終了方法による処理の振り分け」**を実現してみましょう。

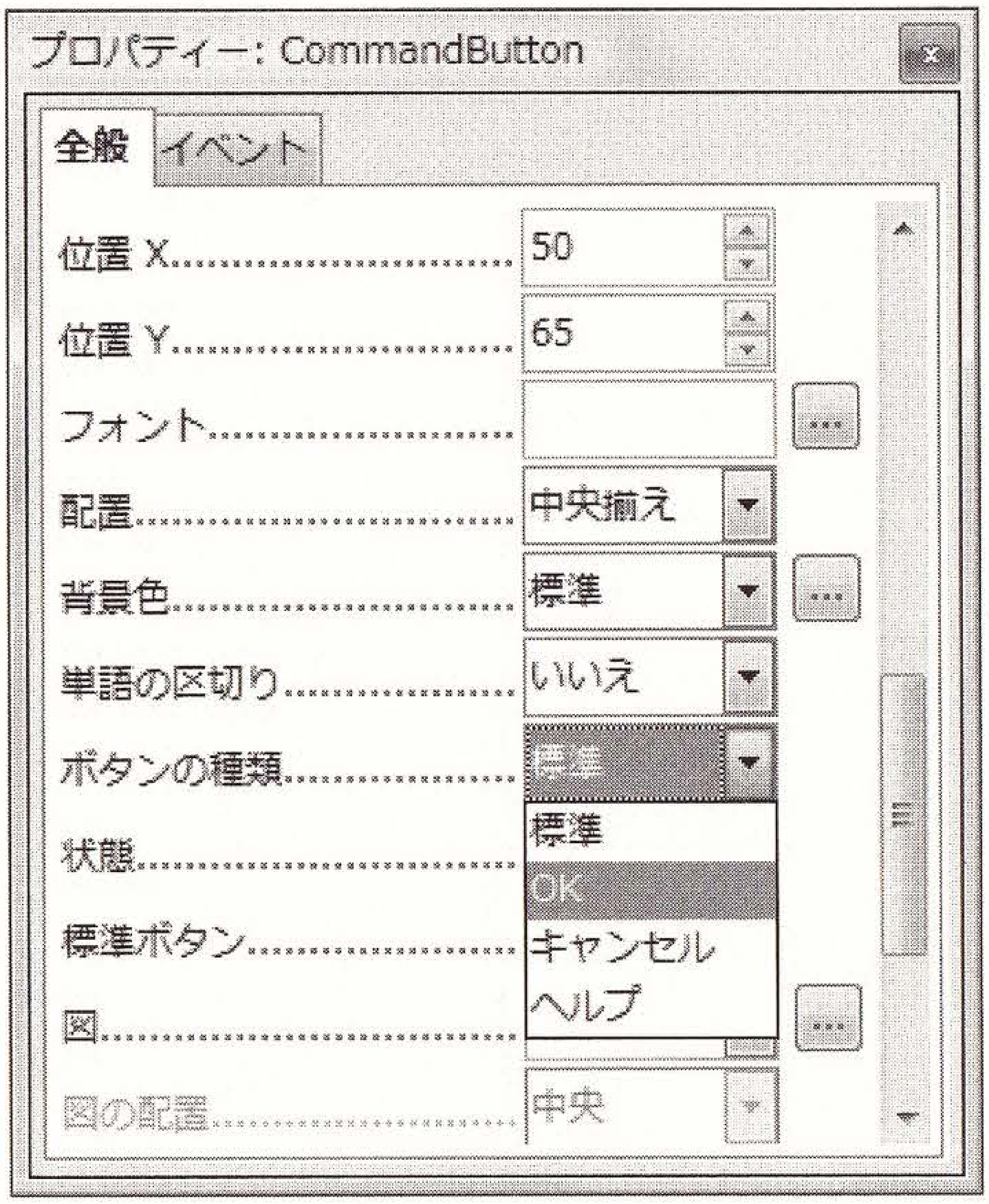
ダイアログに「OK」ボタンを追加するには、まず「ダイアログの管理」画面で先ほど作ったダイアログ「Dialog1」を選んで「編集」ボタンをクリックし、「ダイアログ・エディタ」を呼び出します。

そこでダイアログにツール・ボックスから選んだ「ボタン」を追加してください。



ダイアログにボタンを追加する

ボタンを追加したら、プロパティで「okButton」など、適当な名前を付け、下のほうにある「ボタンの種類」を「OK」にしてください。ついでに、ボタンに表示するタイトルも「決定」など分かりやすいものにしておきましょう。



「ボタンの種類」を「OK」にする

これで「OK」ボタンが追加されたので、以下のマクロでダイアログを表示し、「Execute()」が返す値を見てみます。

```
Sub Main

    Dim dlg As Object

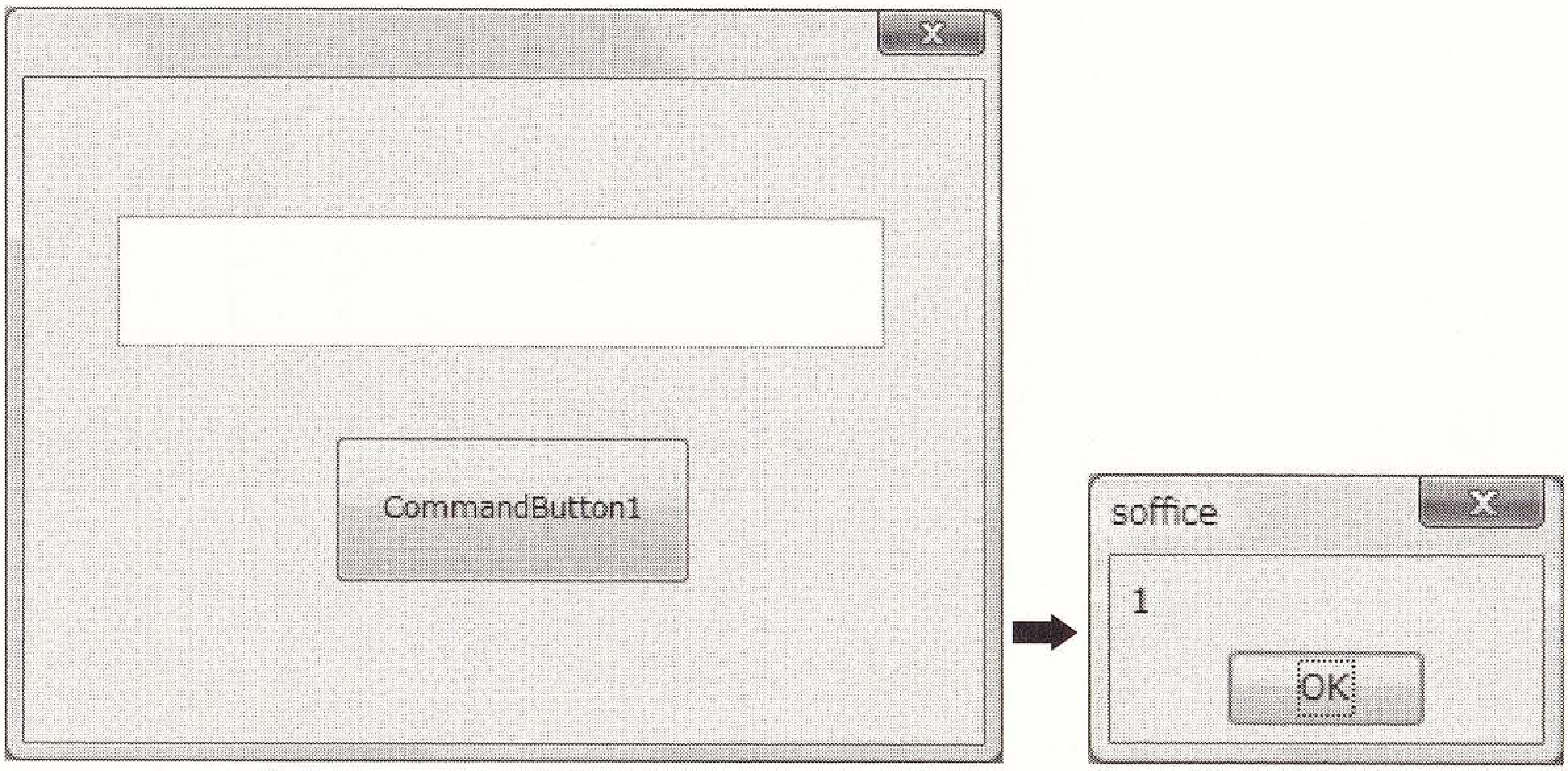
    DialogLibraries.LoadLibrary("Standard")

    dlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1)

    MsgBox dlg.Execute()

End Sub
```

「OK」ボタンをクリックすると、ダイアログが閉じられて「1」が表示され、ダイアログの「×」ボタンなどでダイアログを閉じると「0」が表示されましたね。



終了方法によって、返り値が変わる様子が確認できる

なお、「ダイアログ・エディタ」でボタンの種類を設定する際、「キャンセル」にすると、クリックしたときにキャンセルしてダイアログを閉じる(「Execute( )」が「0」を返す)ボタンを追加することもできます。

3-6 イベント・ハンドラ

「OOo」のマクロでは、**「イベント」**という仕組みを使うことができます。

「イベント」とは、アプリケーション上で文書が開かれたりボタンがクリックされるといった、特定の状況が発生することです。

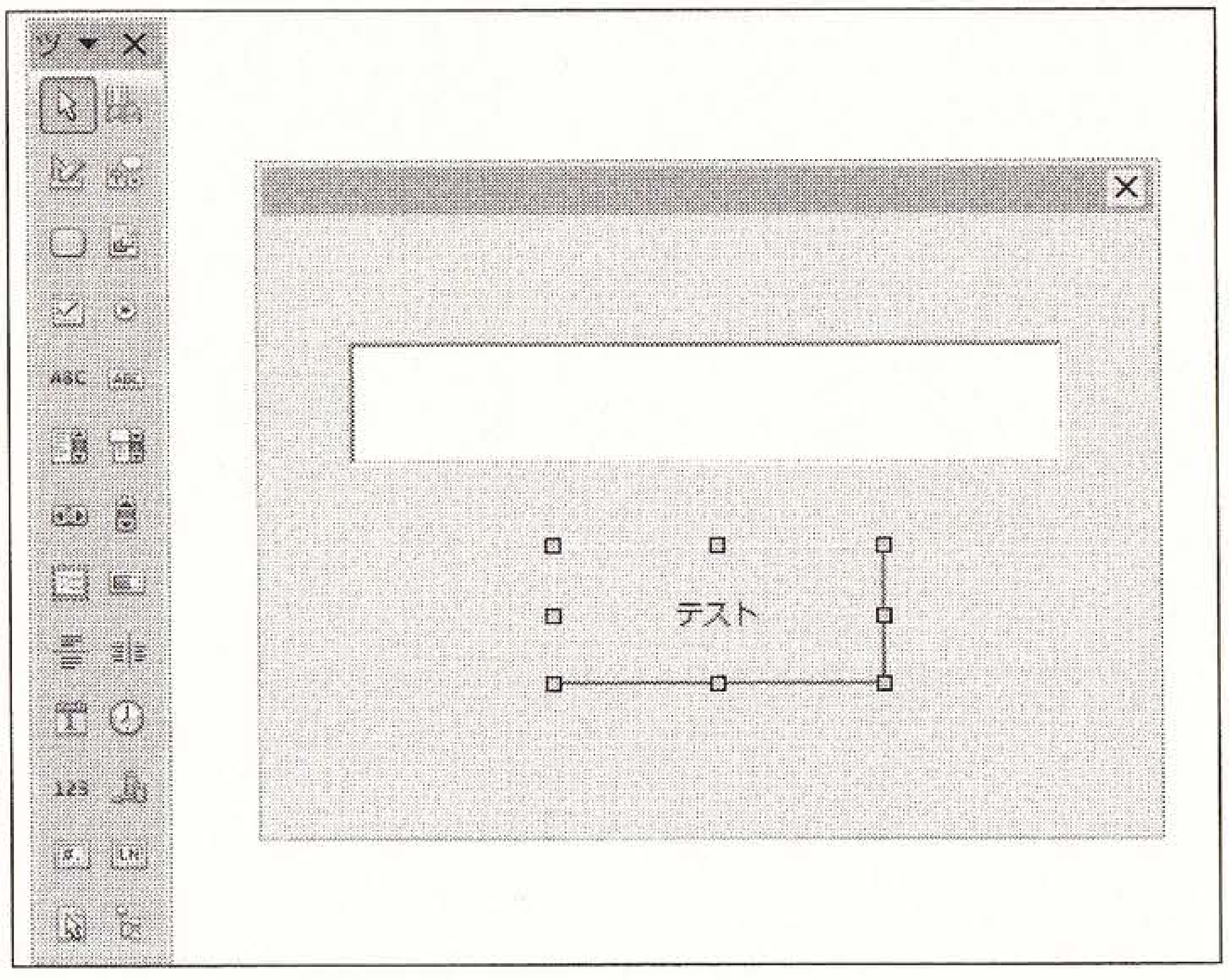
マクロでは、こうした「イベント」が起きたときに実行する**「イベント処理サブルーチン」(イベント・ハンドラ)**を設定し、『ボタンがクリックされたらこの処理を行なう』といった、イベントに応じた処理を行なうことができます。

■「イベント・ハンドラ」の設定

前項で試してみたダイアログでも、GUI関連などでさまざまなイベントが発生します。ここでは、マクロからダイアログのイベントを処理してみて、「イベント・ハンドラ」を設定する基本的な流れを確認してみましょう。

*

まず、前項と同様にダイアログ「Dialog1」を作り、「ダイアログ・エディタ」でテキスト・ボックス「testField」とボタン「testButton」を配置してください。「testButton」のタイトルは、分かりやすいよう「テスト」としておきましょう。



ダイアログの設定

次に、「イベント・ハンドラ」のサブルーチンも含めたマクロを作ります。

今回は、「イベント・ハンドラ」と「Main」という2つの関数でダイアログを処理したいので、ダイアログを参照する変数をサブルーチンの「外」で宣言し、どちらのサブルーチンでも参照可能な「**グローバル変数**」にします。

*

イベント・ハンドラ「EventTest」では、ダイアログの「testText」の「Text」を「イベント・テスト」に設定するようにしてみましょう。

マクロ編集画面で、以下のマクロを入力してください。

```
Dim dlg As Object

Sub Main

    DialogLibraries.LoadLibrary("Standard")

    dlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1)

    dlg.Execute()

End Sub

Sub EventTest

    Dim fld As Object

    fld = dlg.GetControl("testField")

    fld.Text = "イベント・テスト"

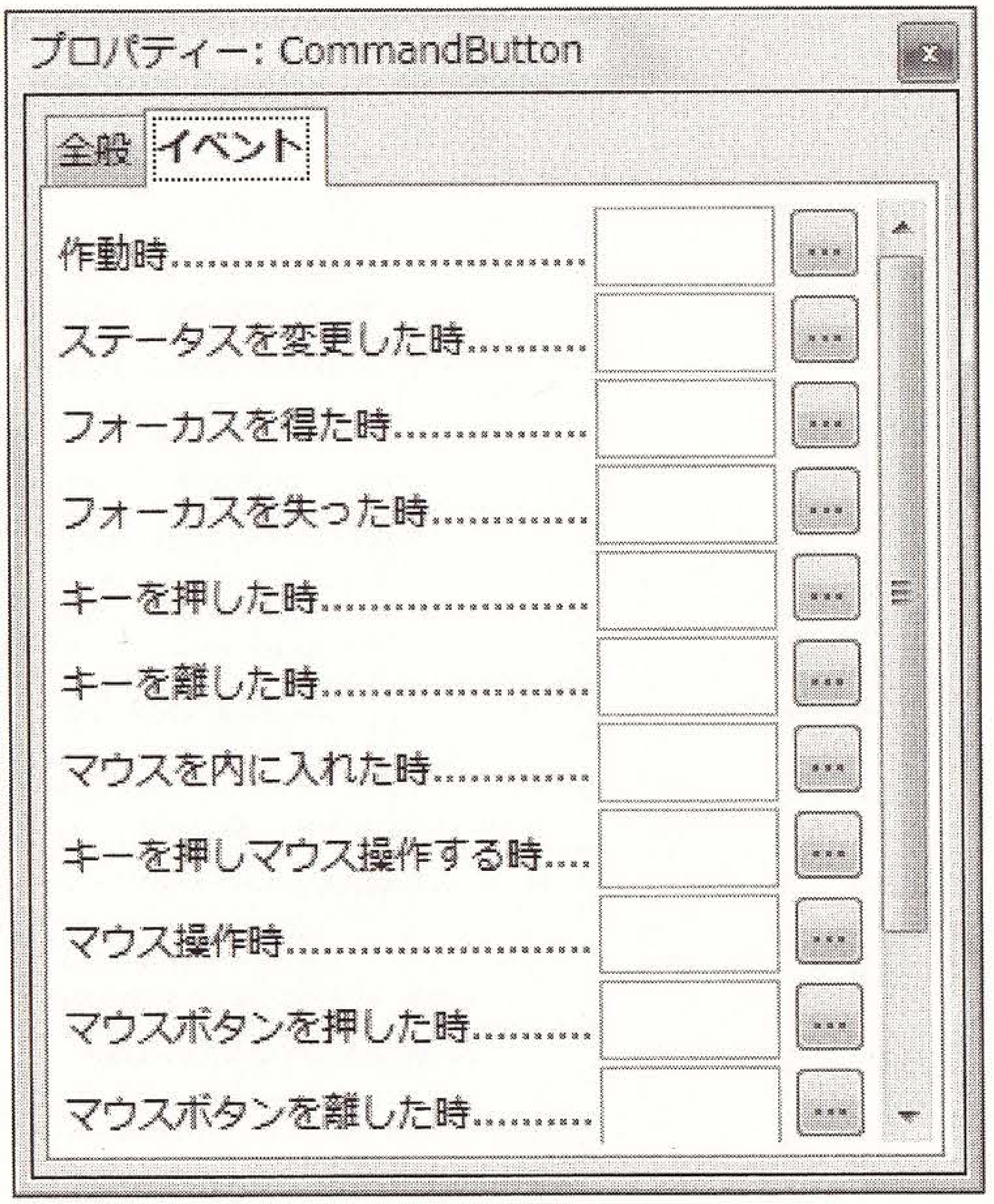
End Sub
```

この状態で実行すると、ダイアログが出てきますが、まだイベントの設定を行なっていないので、ボタンをクリックしても何も起きません。

＊

では、実際に「ダイアログ・エディタ」でボタン「testButton」のイベント・ハンドラに「EventTest」を割り当てましょう。

ボタンの「プロパティ設定」画面で「イベント」タブを選び、「イベントの設定」ページに移ってください。

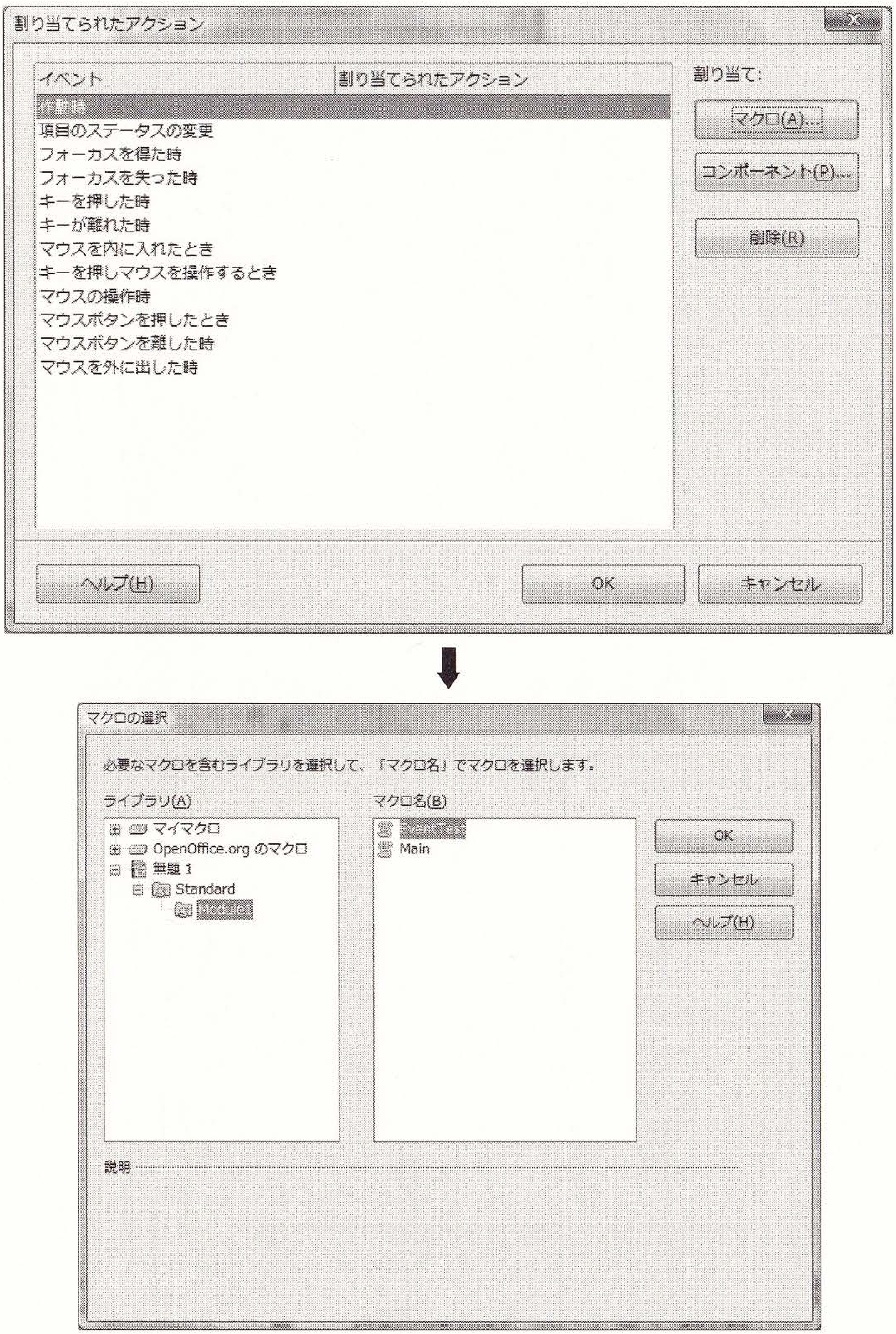


「イベントの設定」ページ

「作動時」「ステータスを変更した時」など、多数のイベントが列挙されているので、「作動時」の設定ボタン(…)をクリックします。

すると、「イベントに割り当てるアクション」の設定画面になるので、「マクロ」を選び、先ほど入力したマクロを選択します。

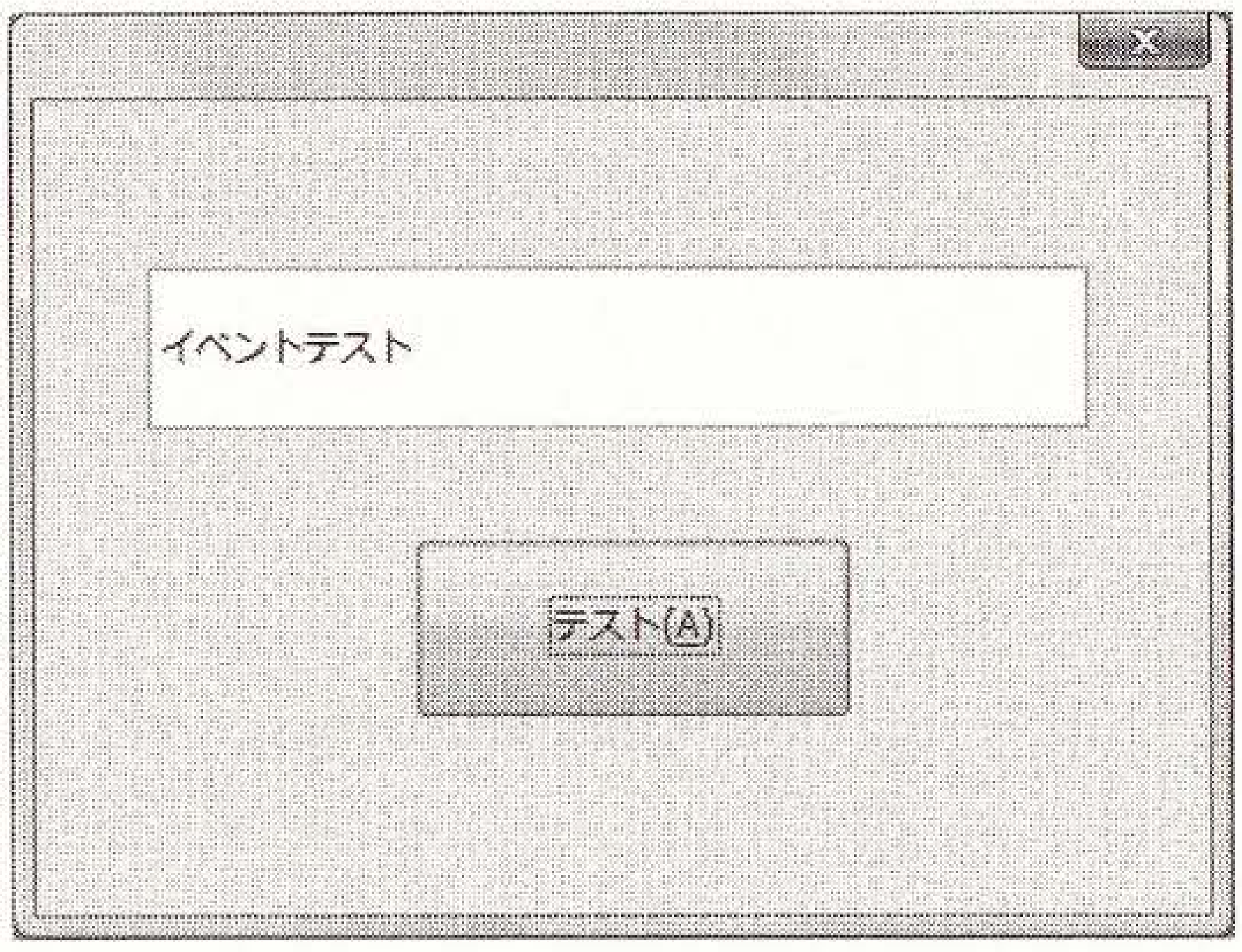
マクロを選択すると、「マクロ名」にサブルーチンの一覧(「Main」と「EventTest」)が出てくるので、「EventTest」を選んでください。



「イベントに割り当てるアクション」の設定画面

「OK」ボタンをクリックしてイベントの設定を終えたら、マクロを実行してみましょう。

こんどは、ボタンをクリックすると、「作動時」イベント発生時に実行するよう設定しておいたサブルーチン「TestEvent( )」が実行され、テキスト・ボックスに「イベントテスト」の文字列が入るはずです。



テキスト・ボックスに文字列が入力される

3-7 XMLの処理

「データベース・システム」や「OOoの保存ファイル」など、「**XML**」を利用する機会が増えてきました。
「OOo」のAPIにはXMLに関する機能も用意されているので、マクロからこれらのAPIを呼び出すことで、XMLファイルで蓄積されているデータを活用できるようになります。

■XMLファイルを読み込み、内容を取得する

まずは試しに、XMLファイルを読み込んで、内容を読み取ってみましょう。

XMLファイルを扱う際は、「**DOM**」というXMLの文書構造にアクセスする仕組みが利用されます。

「DOM」には、XMLの「要素を検索」や「内容や属性の値を取得」など、XMLを処理するための機能が一通り用意されているので、何らかの手段で処理対象XMLファイルにアクセスできる「DOMオブジェクト」(あるいはインターフェイス)を取得すれば、後はその「DOM」を通してXMLを処理できるわけです。

「DOM」は、「OOo」だけでなく、「JavaScript」や「Java」「各種XMLライブラリ」でも用意されているので、すでに他の環境で「DOM」を利用したことがある、という人なら、XMLから「DOM」を得る方法さえ把握すれば、すぐに「OOo」のマクロでもXMLを活用できるでしょう。

■「DOM」の取得

「OOo」のマクロでは、このDomを「**DocumentBuilde**」サービスから取得できます。

まず、「DocumentBuilde」サービスを作り、続いて「**parseURI( )**」にXMLの場所を渡せば、「DOMのインターフェイス」(「**XDocument**」インターフェイス)をもつXMLの文書オブジェクトを取得できます。

たとえば、「d:¥test.xml」を扱うDOMは、

```
builder = CreateUnoService("com.sun.star.xml.dom.DocumentBuilder")


doc = builder.parseURI("file:///d:/test.xml")
```

のようにして取得できます。

後は、取得したオブジェクトに対して、「XDocument」インターフェイスの機能を使ってアクセスしながら情報を取得していくわけです。

今回は、以下のようなXMLを読み込んでみましょう。

```
<?xml version="1.0"?>
<itemlist>
    <item>
        <name>商品1</name>
        <price>1000</price>
    </item>
    <item>
        <name>商品2</name>
        <price>2000</price>
    </item>
    <item>
        <name>商品3</name>
        <price>3000</price>
    </item>
</itemlist>
```

このXMLを「d:¥test.xml」に配置し、「item」の情報をDOMから取得してみます(保存時に文字コードを指定できるエディタで、「UTF-8N」や「UTF-8」で保存して、試してみてください)。

「test.xml」を配置したら、その場所を「parseURI( )」の引数に渡して、以下のマクロを実行してみてください。

```
Sub Main

    Dim builder As Object
    Dim doc As Object
    Dim items As Object
    Dim itemNum As Long
    Dim itemName As String
    Dim itemPrice As String
```

```
    builder = CreateUnoService("com.sun.star.xml.dom.Document
Builder")

    ' XMLオブジェクト取得
    doc = builder.parseURI("file:///d:/test.xml")

    ' XMLのitem要素の集合を取得
    items = doc.getElementsByTagName("item")

    ' itemの数を取得
    itemNum = items.getLength()

    ' 各itemの情報を表示
    For i = 0 To itemNum - 1

        itemName = items.item(i).getElementsByTagName("name").
item(0).getFirstChild().getNodeValue()
        itemPrice = items.item(i).getElementsByTagName("price").
item(0).getFirstChild().getNodeValue()

        MsgBox itemName & ":" & itemPrice

    Next

End Sub
```



実行結果

このマクロでは、XMLオブジェクト「**doc**」から、指定の名前をもつ要素を列挙する「**getElementsByTagName( )**」で「item」の集合を取得し、各「item」の「name」と「price」を表示するようにしてみました。

＊

「parseURI( )」に指定するURIはネット上のURLも指定できるので、Webサーバに保存してある「XML文書」や「XMLデータ」を返す「Webサービス」などの「XML文書」を読み込むこともできます。

MEMO

第4章

マクロとアプリケーションの連携

前章までで、「OOo」のマクロに関する基本的な部分を一通り見てきました。
　ここからは実際の「OOo文書」に組み込むマクロで、「OOo」の各アプリケーションの機能を活用してみましょう。

4-1 Calcの表計算シートの扱い方

表計算ソフト「Calc」の機能をマクロから利用してみることにします。

マクロから各アプリケーションの機能を利用するには、まず「**ThisComponent文**」でドキュメントのオブジェクトを取得します。

このドキュメントにアクセスし、「Calc」の表計算シートや「Base」のデータベースなど、各アプリケーションがもっている機能を利用するわけです。

■マクロから「Calc」の表計算ドキュメントを取得する

まずはマクロで「表計算ドキュメント」を取得しましょう。

「Calc」を起動したら、「マクロの編集」画面で以下のようなマクロを実行し、「ThisComponent」で取得するドキュメントで、どんなインターフェイスやサービスがサポートされているか見てみることにします。

```
Sub Main

    Dim doc As Object
    Dim serviceList As Object
    Dim num As Long
    Dim i As Long

    doc = ThisComponent

    serviceList = doc.getSupportedServiceNames()

    num = UBound(serviceList) - LBound(serviceList) + 1

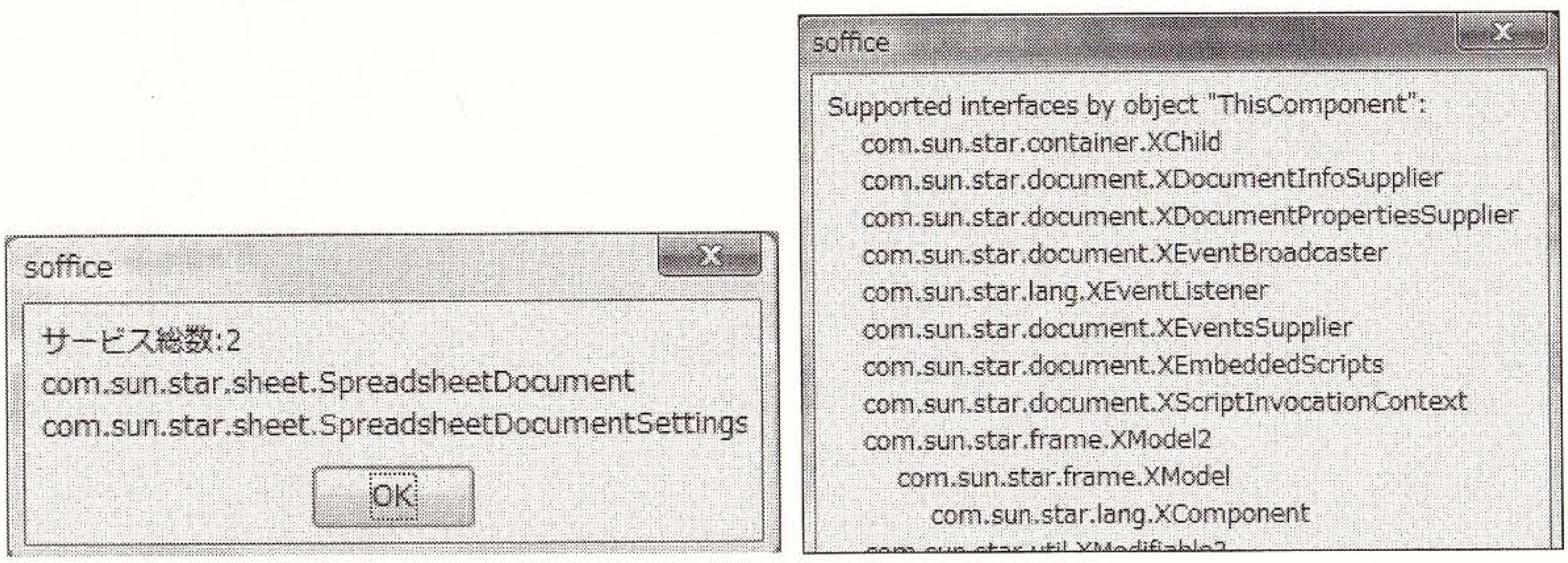
    info = "サービス総数:" & num &  Chr(10)
```

```
    For i = LBound(serviceList) to UBound(serviceList)
        info = info & serviceList(i) & Chr(10)
    Next

    MsgBox info

    MsgBox doc.Dbg_SupportedInterfaces

End Sub
```



「Calc」でサポートされているサービスとインターフェイス

実行してみると、(自分自身の他に)、

```
com.sun.star.sheet.SpreadsheetDocument
com.sun.star.sheet.SpreadsheetDocumentSettings
```

という2つのサービスがあり、それ以外にも多数のインターフェイスをもっているようです。

■「コレクション」の取得

「SpreadsheetDocument」サービスの「XSpreadsheetDocument」インターフェイスには、ドキュメント内のシートをまとめた**「コレクション」(集合)**を取得する「getSheets( )」メソッドがあるので、とりあえずこれでドキュメントに含まれるシートのコレクションを取得してみましょう。

シートのコレクションを取得すると、マクロ側で「( )」の中に「0」から始まる「インデックス」や「シートの名前」を入れて、コレクション内のシートにアクセスできます。

たとえば、ドキュメントに含まれる最初のシートを取得するなら、

```
ThisComponent.getSheets(0)
```

とするわけです。これは、

```
ThisComponent.Sheets(0)
```

と書くこともできます。

以下のマクロを実行してみると、ドキュメントに含まれる最初のシートを取得して、その名前を表示できます。

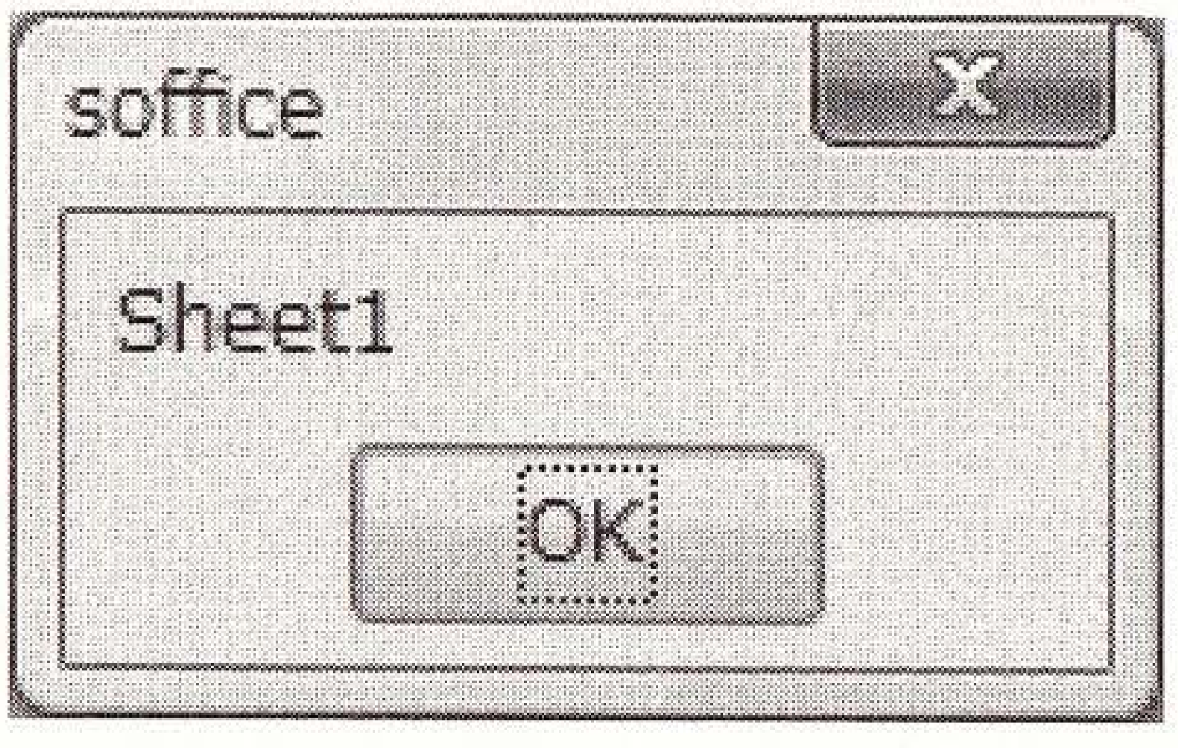
```
Sub Main

    Dim firstSheet As Object

    firstSheet = ThisComponent.Sheets(0)

    MsgBox firstSheet.Name

End Sub
```



最初のシートのシート名が表示される

＊

また、「Sheets」には、ある名前のシートがあるかを調べる「**hasByName( )**」メソッドや、シートのコレクションに新しいシートを挿入(作成)する「**insertNewByName( )**」、指定した名前のシートを削除する「**removeByName( )**」といったメソッドもあります。

「insertNewByName( )」の引数には、作るシートの「名前」と「位置」(「0」からはじまるインデックス。指定インデックスの前に挿入される)を、他のメソッドの引数には、シートの「名前」を指定します。

ただし、「insertNewByName( )」でシートを作る場合は、すでに同名のシートがあるとエラーになるので、作る前に同じ名前のシートがないか確認するようにしましょう。

たとえば、新しいシートを「testSheet41」という名前で作るなら、

```
Sub Main

    Dim sheets As Object
    Dim sheetName As String

    sheetName = "testSheet41"

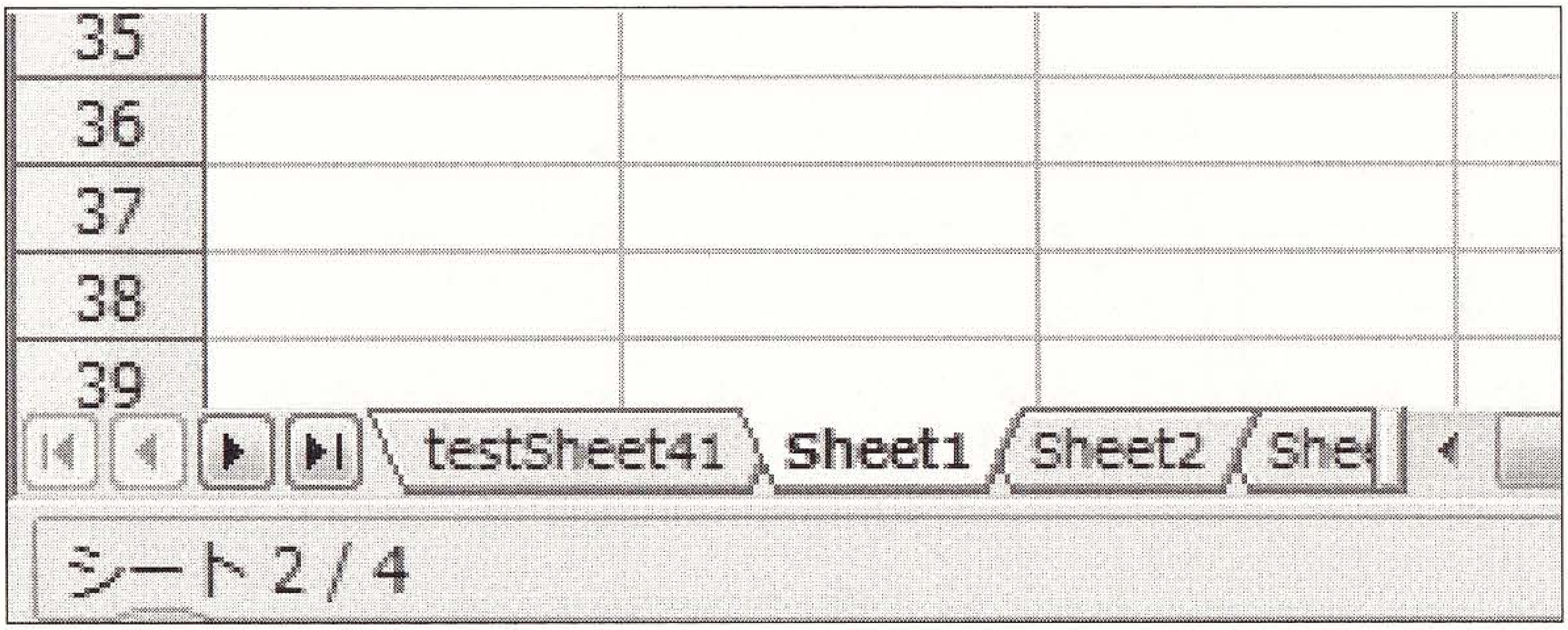
    sheets = ThisComponent.Sheets

    If sheets.hasByName(sheetName) Then
        sheets.removeByName(sheetName)
    End If

    sheets.insertNewByName(sheetName, 0)

End Sub
```

のようにします。



指定した新しいシートが挿入される

＊

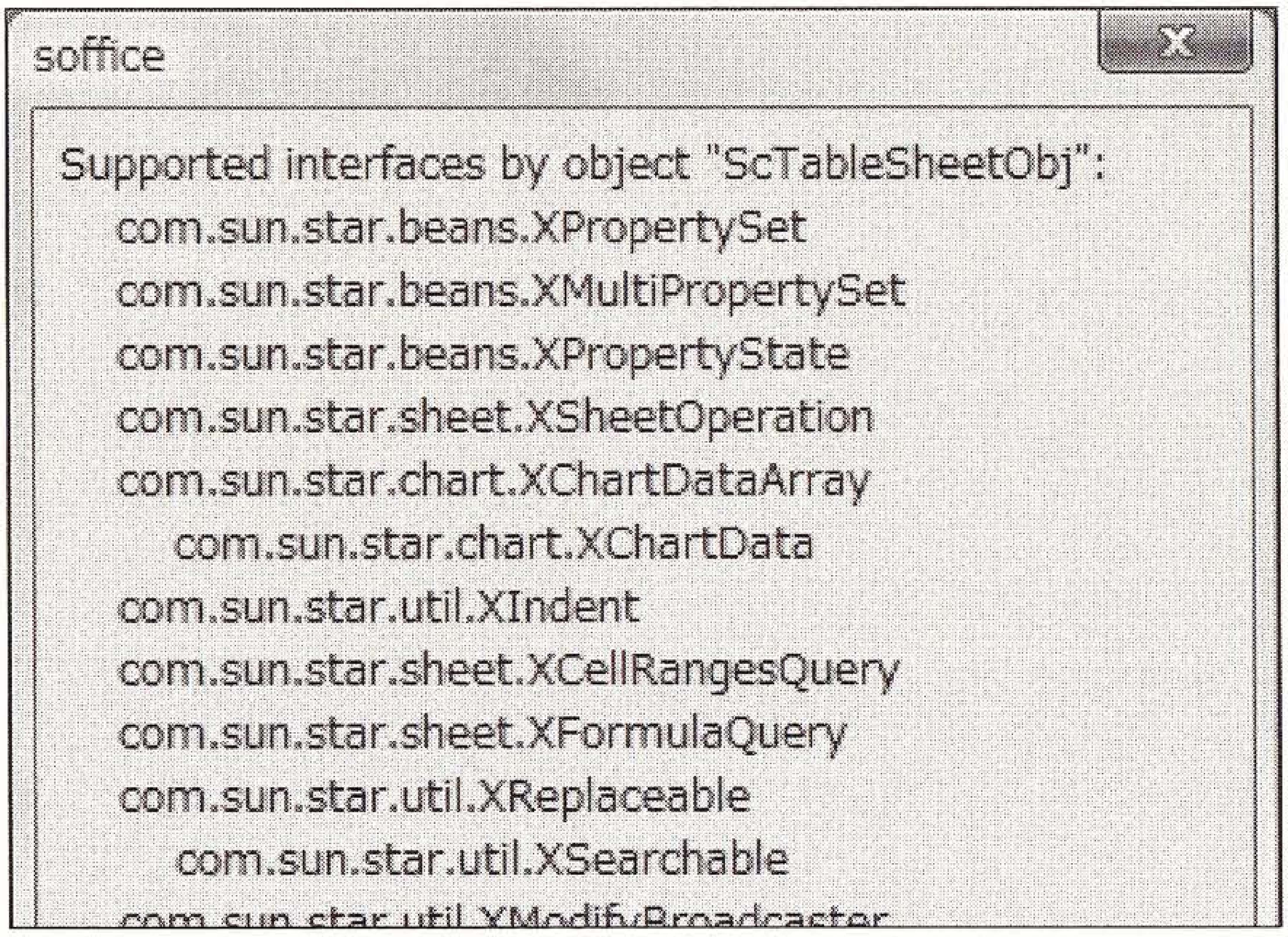
個々のシートについても、どんなインターフェイスをもっているか見てみましょう。

```
MsgBox firstSheet.Name
```

の部分を、

```
MsgBox firstSheet.Dbg_SupportedInterfaces
```

として実行してみてください。シートも非常に多機能で、たくさんのインターフェイスをもっているみたいですね。



シートがもっているインターフェイスの一部

■セルにアクセスする

シートを取得したら、「**XCellRange**」インターフェイスでセルにアクセスできます。

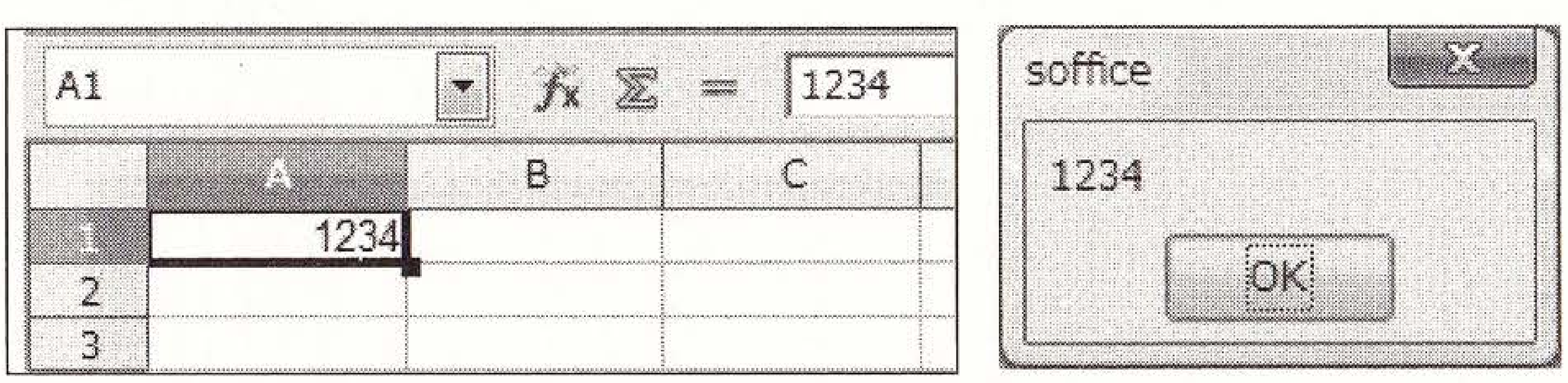
「XCellRange」は、シート中の一定の範囲のセルをまとめたもので、「**getCellByPosition( )**」で位置を指定すると、その位置にあるセルを取得できます。

「getCellByPosition( )」の位置指定は、最初の引数に「A,B,C」といった「列」(横方向)の位置を、2番目の引数に「行」(縦方向)を、いずれも「0」から始まるインデックスで指定します。

＊

試しに、シートの左上端「A1」セルに適当な数字を入れてから、以下のマクロを実行してみてください。

```
Sub Main
    MsgBox ThisComponent.Sheets(0).getCellByPosition(0, 0).Value
End Sub
```



実行結果

「getCellByPosition( )」で返されるセルの「Dbg_SupportedInterfaces」を見ると、セルも実にたくさんのインターフェイスにアクセスできるようです。

＊

以上で、「Calcドキュメント」「ドキュメントにあるシートのコレクション」「個々のシート」「セル」といった「Calcの各要素」(オブジェクト)にマクロからアクセスする方法を一通り確認できました。

4-2 セルの扱い方

セルの扱いについて、もう少し詳しく試してみましょう。

4-1節では、「getCellByPosition( )」で指定セルを取得し、その「Value」プロパティを参照してみましたが、他にもさまざまな「プロパティ」や「メソッド」にアクセスできます。

ここでは、セルの「背景」や「表示位置」(「右寄せ」「中央」「左寄せ」)といった表示設定を変えてみましょう。セルには、セルの「プロパティ」をまとめて扱う「**com.sun.star.table.CellProperties**」サービスが含まれているので、このサービスのプロパティを変更してみます。

■背景色の変更

最初に、「セルの背景色」を変えてみましょう。

セルの背景色は、「CellProperties」の「**CellBackColor**」プロパティで設定できます。

このプロパティに設定する値は、最初の2桁が「赤」、次の2桁が「緑」、最後の2桁が「赤」の、合計6桁の16進数です。

たとえば、以下のようにすると、最初のシートの「A1」セルの背景を赤くすることができます。マクロを実行したら、アプリケーションのウインドウを表示して実際に「A1」セルが赤くなっているのを確認しましょう。

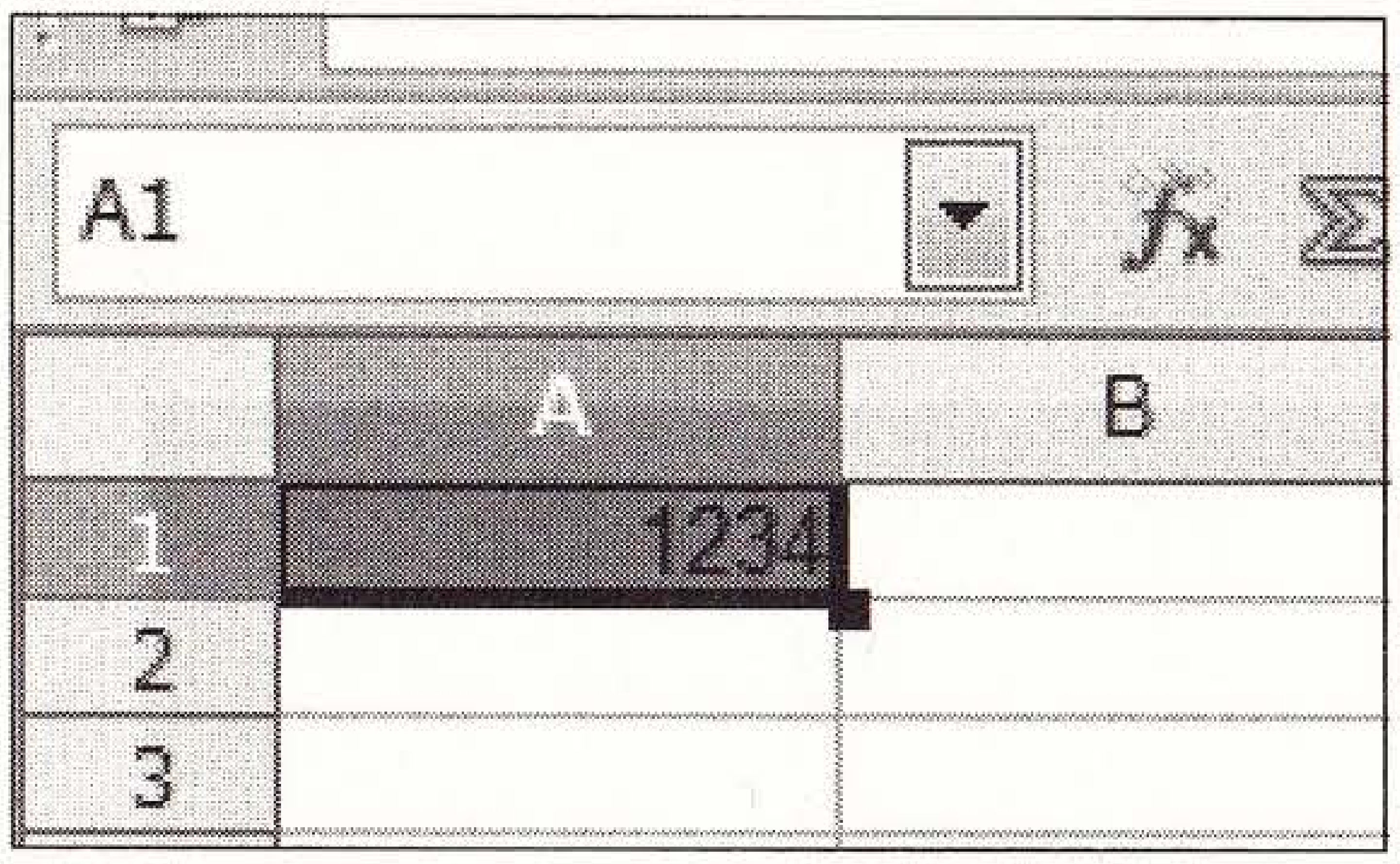
```
Sub Main

    Dim testCell As Object

    testCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)
```

```
    testCell.CellBackColor = &HFF0000

End Sub
```



指定のセルの背景色が変更される

続いて、

```
testCell.CellBackColor = &HFF0000
```

の部分を、

```
testCell.CellBackColor = &H0000FF
```

として青くしてみたり、

```
testCell.CellBackColor = &H00FF00
```

として緑にしてみたりして、16進数6桁による色の指定を試してみましょう。

■文字列の表示位置を変更する

続いて、セルの中で「**文字列をセンタリング**」してみます。

セル内部のセンタリングは、「CellProperties」にある「**HoriJustify**」(横方向)、「**VertJustify**」(縦方向)という2つのプロパティで設定可能です。

設定可能な値は、「HoriJustify」が「com.sun.star.table.CellHoriJustify」で定義されている、

com.sun.star.table.CellHoriJustify.STANDARD	……デフォルト
com.sun.star.table.CellHoriJustify.LEFT	……………左寄せ
com.sun.star.table.CellHoriJustify.CENTER	………中央寄せ
com.sun.star.table.CellHoriJustify.RIGHT	…………右寄せ

「VertJustify」が「com.sun.star.table.CellVertJustify」で定義されている、

com.sun.star.table.CellVertJustify.STANDARD	……デフォルト
com.sun.star.table.CellVertJustify.TOP	………………上寄せ
com.sun.star.table.CellVertJustify.CENTER	………中央寄せ
com.sun.star.table.CellVertJustify.BOTTOM	………下寄せ

となっています。

たとえば、「A1」セルを「横方向中央寄せ」にするなら、

```
Sub Main

    Dim testCell As Object

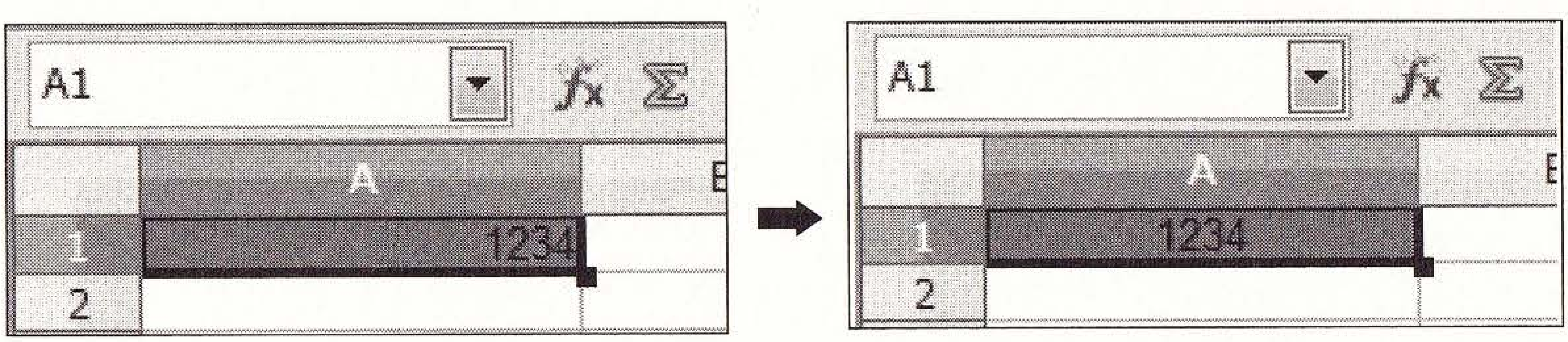
    testCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)

    testCell.HoriJustify = com.sun.star.table.CellHoriJustify.CENTER

End Sub
```

とするわけです。

「A1」セルに何か文字列を入れた状態でマクロを実行し、実際に表示が変わることを確認してみましょう。



セル内のデータが「横方向中央寄せ」に変わる

■文字の属性を変更する

「CellProperties」に続いて、「**CharacterProperties**」でセルの「文字の属性」を変えてみましょう。

「CharacterProperties」には、文字の「色」や「サイズ」「下線」など各種属性を表わすプロパティがあるので、それらを変更することで、セルの文字に変化を与えることができます。

たとえば、文字の「色」を変えるなら、以下のように「**CharColor**」を変更します。色の指定は、セルの背景同様に、RGBそれぞれ2桁で、計6桁の16進数です。

```
testCell.CharColor = &H808080
```

文字列に「下線」(アンダーライン)を引くには、「**CharUnderline**」に「**com.sun.star.awt.FontUnderline**」で定義されている定数を指定します。

通常は、「SINGLE」(実線)や「DOTTED」(点線)を使うことになるでしょう。

*

以下のマクロを実行すると、「B2」セルの背景を「水色」にし、文字の色を「赤」にして「下線」が付くようになります。

```
Sub Main

    Dim testCell As Object

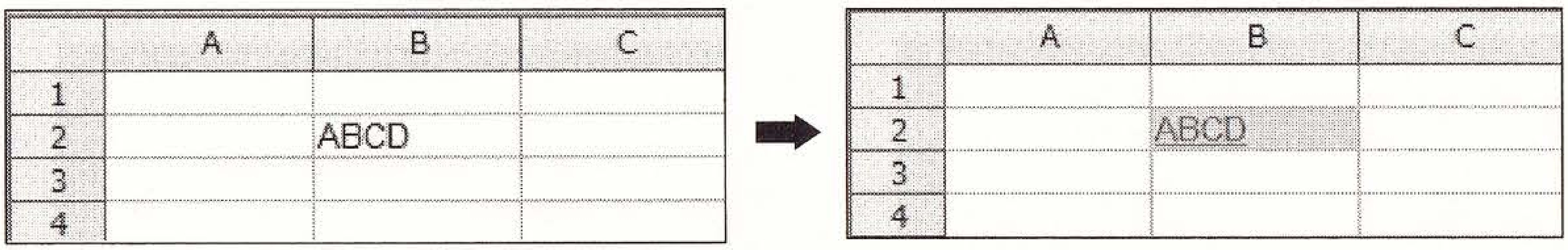
    testCell = ThisComponent.Sheets(0).getCellByPosition(1, 1)

    testCell.CellBackColor = &H00FFFF

    testCell.CharColor = &HFF0000

    testCell.CharUnderline = com.sun.star.awt.FontUnderline.SINGLE
```

```
End Sub
```



セルの「背景色」と「属性」が、それぞれ変更される

■シート全体に変更をかけることも可能

これまで見てきた「CellProperties」や「CharacterProperties」は、セルの集合である「CellRange」に対しても有効です。

「Sheets(0)」で取得したシートは、(シートのセル全体を保持する)「CellRange」としても扱えるので、以下のようにすると、シート全体の「セル」や「文字の属性」を変更できます。

```
Sub Main

    Dim testCells As Object

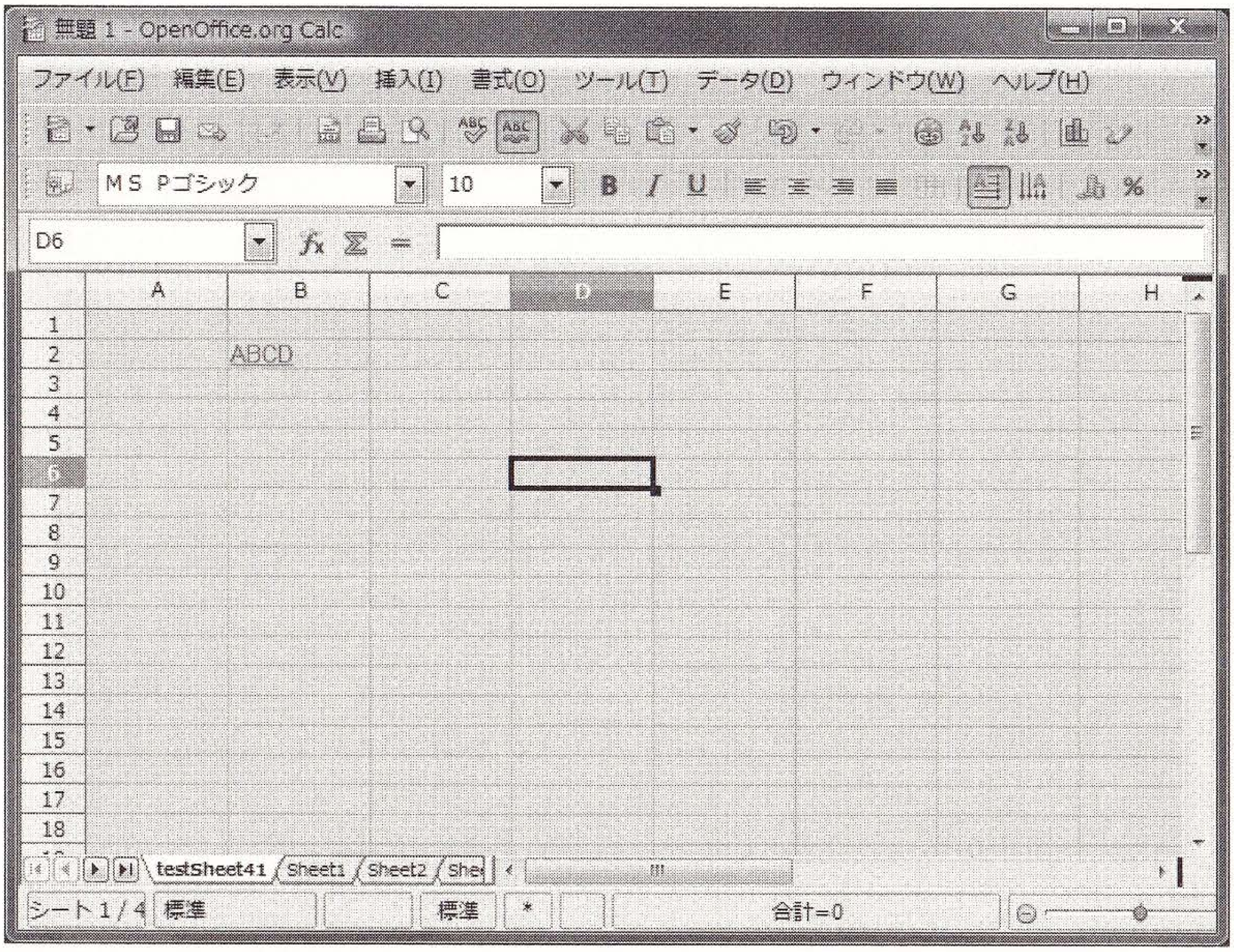
    testCells = ThisComponent.Sheets(0)

    testCells.CellBackColor = &H00FFFF

    testCells.CharColor = &HFF0000

    testCells.CharUnderline = com.sun.star.awt.FontUnderline.SINGLE

End Sub
```



シート全体の情報を変更することもできる

MEMO

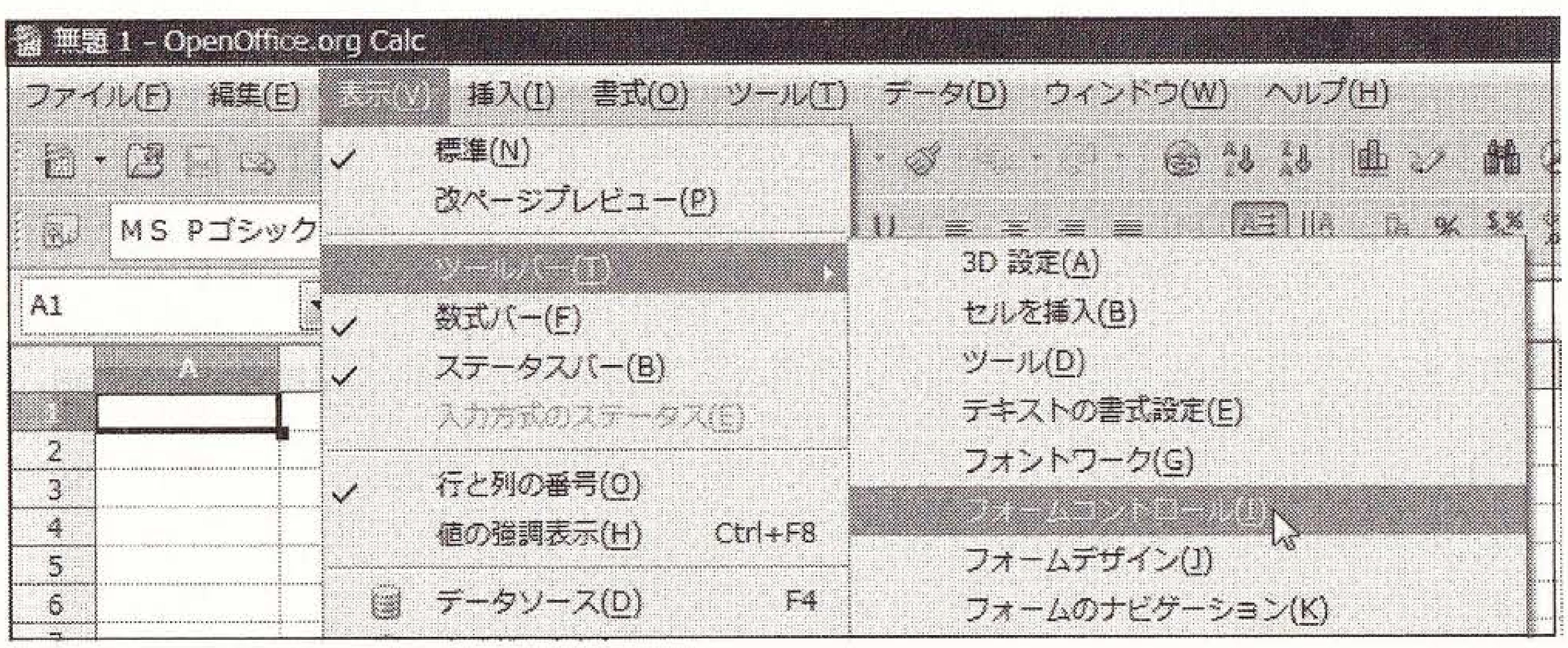
4-3 CalcのシートにGUI部品を配置

第3章でダイアログにボタンを追加し、ボタンがクリックされたときに実行するイベント・ハンドラのマクロを作ってみました。

「Calc」のシートにも、同様にボタンなどのGUI部品を配置し、イベントに応じて実行するマクロを設定することが可能です。

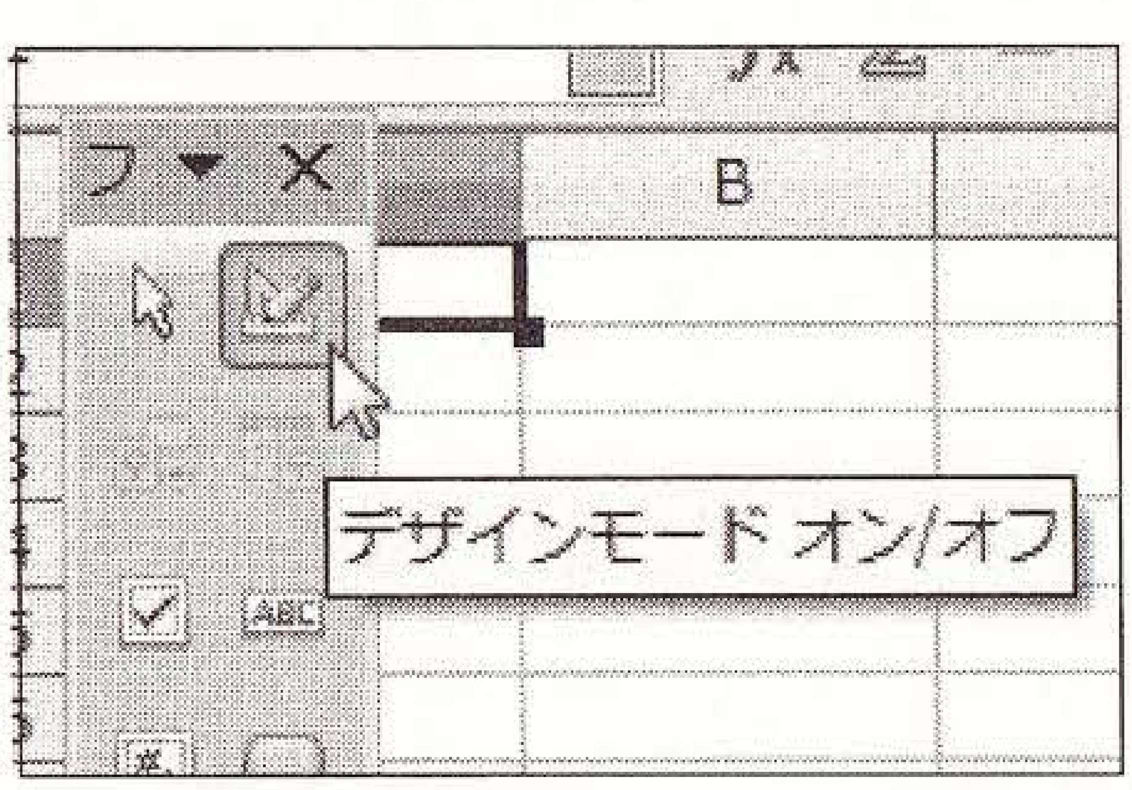
■シートにGUI部品を配置する

シートにGUI部品を配置するには、フォーム・コントロールのツールバーを使います。「表示」メニューの「ツールバー」で、「フォームコントロール」にチェックを入れて表示させてください。



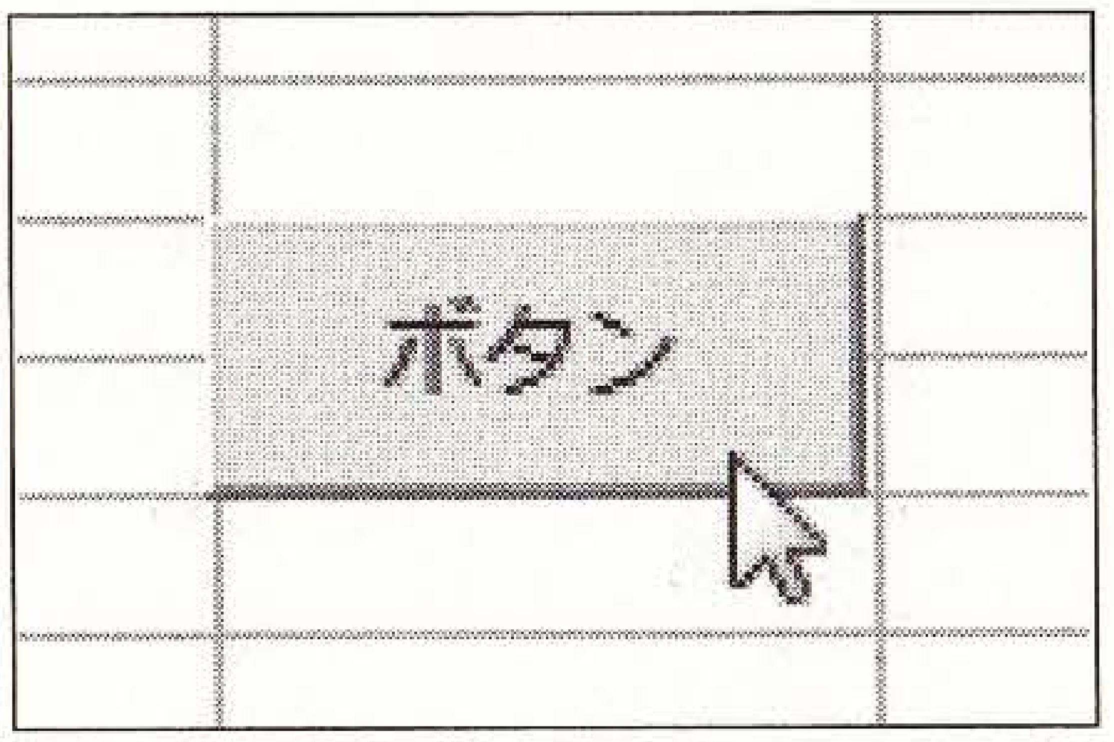
「フォームコントロール」を表示

ツールバーが表示されたら、まず、「デザインモード」がオンになっている(ツールバーのGUI部品が選択できる)状態になっていることを確認します。



「デザインモード」は、オンでGUI部品の編集をし、オフで実際の動作確認

そして、後はダイアログのときと同様に、「ボタン」を選んでドラッグして配置しましょう。



ボタンが配置された

■ボタン用のマクロを作る

ボタンを配置したら、ボタンをクリックするときに実行するテスト用のマクロ・サブルーチンを定義します。

マクロの編集画面で、以下のようなテスト・サブルーチンを定義してみてください。

```
Sub btnTest
    MsgBox "ボタンクリックテスト"
End Sub
```

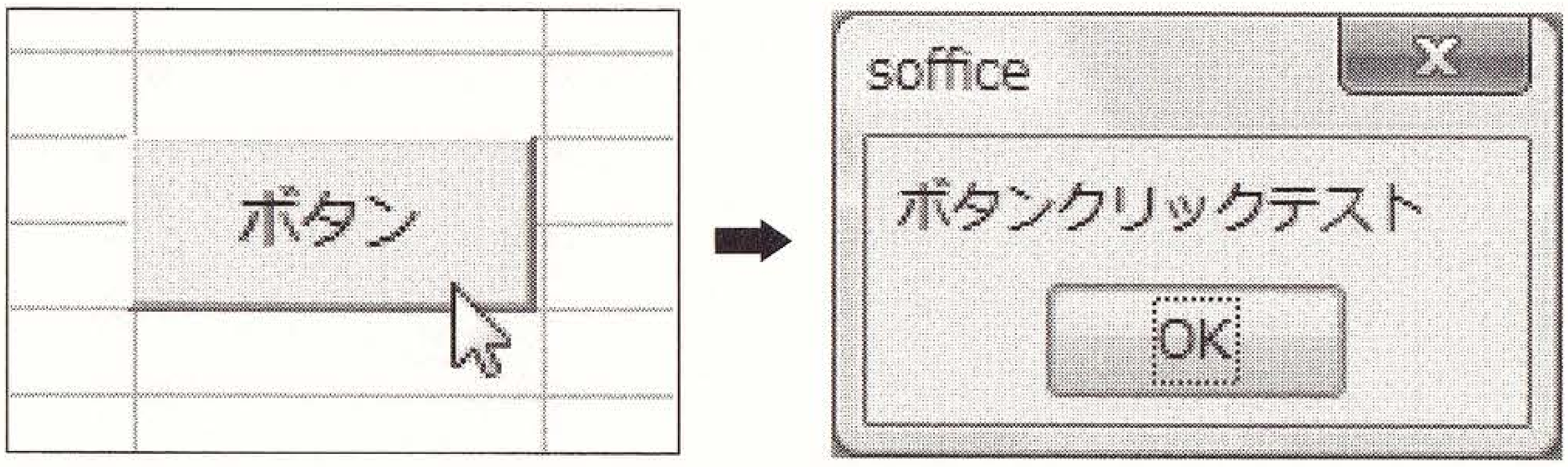
■属性の設定

続いて、シート上のボタンを左クリックし、「コントロール」で属性の設定に入ります。

ここでは、ダイアログのときと同じく、「イベント」の「作動時」に「マクロ」の割り当てを行ない、先ほど定義した「btnTest」を指定してください。

「OK」をクリックして設定を完了し、「コントロールの属性」設定ダイアログを閉じたら、「デザインモード」をオフにして、ボタンをクリックしてみましょう。

先ほど割り当てたマクロが実行されて、メッセージが表示されたと思います。



ボタンをクリックすると、メッセージが表示される

＊

今回は、ボタンの作動時のイベント設定を試してみましたが、GUI部品のイベントには、他にも「フォーカス移動時」や「マウス関連」のイベントなど、さまざまな種類のイベントがあります。

マウスの動きに応じて表示を変えてみたり、フォーカス移動時に自動的に内容を更新したりと、イベントに応じてマクロの処理を実装することで、表計算シートを高機能化してみてください。

4-4 Calcのシートに図形を配置

「Calc」のシートには、「**DrawPage**」という各種の図形を配置するための領域があります。「図形オブジェクト」を生成し、「DrawPage」に置くことで、「Calc」のシートに、各種の図形を表示できるわけです。

■図形の配置と、「位置」「大きさ」の設定

シートの「DrawPage」に図形を配置するには、以下のようにします。

[1]「createInstance」で「図形オブジェクト」を生成。
[2]「図形オブジェクト」の「位置」「大きさ」「色」などを設定。
[3]シートの「DrawPage」に「図形オブジェクト」を追加。

「DrawPage」に配置する図形は、「**Shape**」と呼ばれ、「**com.sun.star.drawing**」以下にまとめられています。

実際にオブジェクトを作る際には、ドキュメントの「**createInstance**」に作りたい「Shape」の「名前」を渡すと「オブジェクト」が作られるので、「Object型」変数に格納しましょう。

＊

たとえば、ドキュメントの最初のシートの「DrawPage」に、「長方形」(RectangleShape)を追加するには、以下のようにします。

```
Sub Main

    Dim objShape As Object
    Dim ptShape As New com.sun.star.awt.Point
    Dim sShape As New com.sun.star.awt.Size

    objShape = ThisComponent.createInstance("com.sun.star.
drawing.RectangleShape")

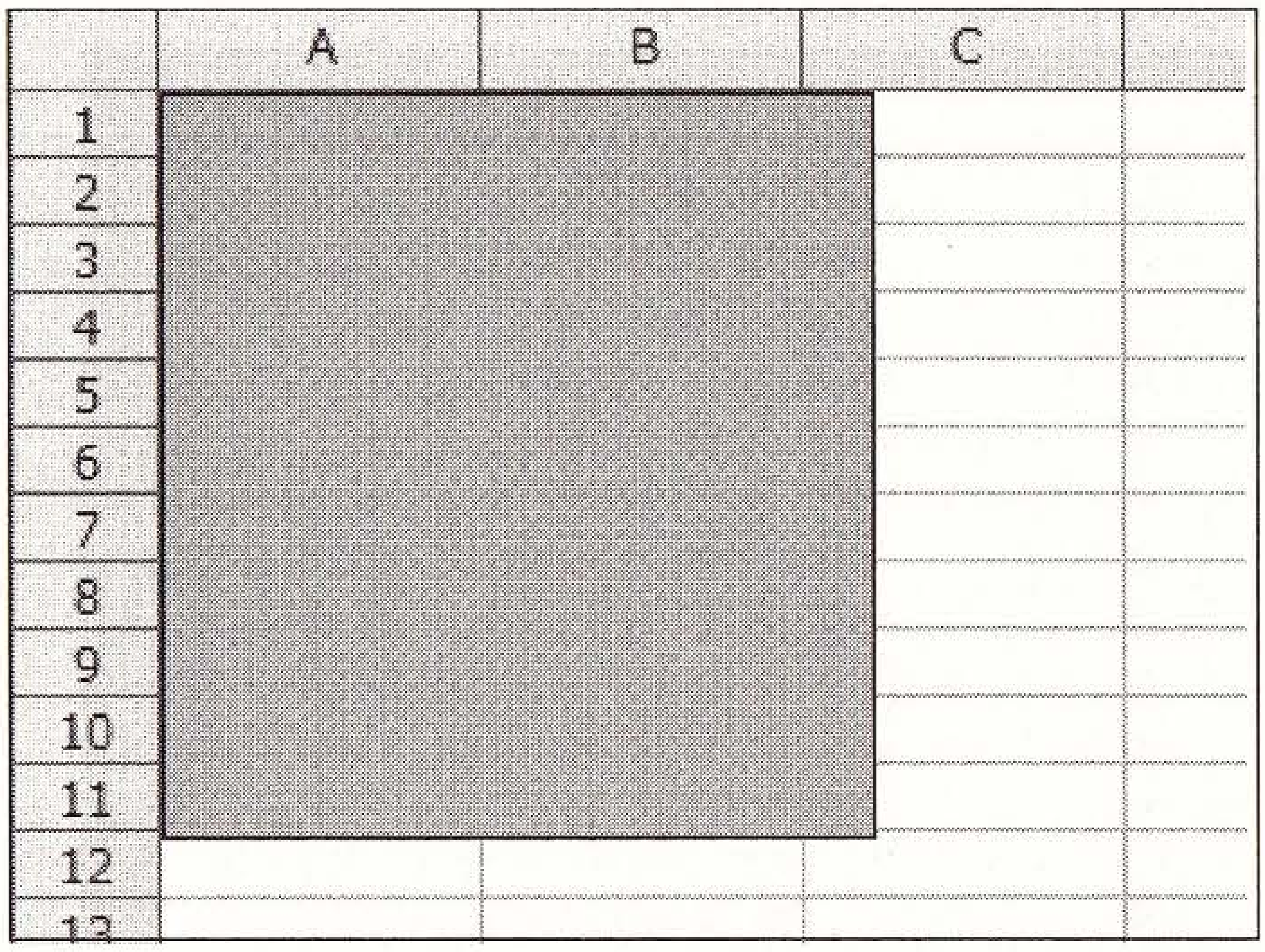
    ptShape.X = 0
    ptShape.Y = 0

    sShape.Width = 5000
    sShape.Height = 5000

    objShape.setPosition(ptShape)
    objShape.setSize(sShape)

    ThisComponent.Sheets(0).DrawPage.Add(objShape)

End Sub
```



正方形が配置される

*

「位置」や「大きさ」の設定は、**「Point」「Size」**というオブジェクトを作って(New)おいて値を設定し、「Shape」に渡す形で行ないます。

「位置」を設定する「Point」の「X」「Y」、「大きさ」を設定する「Size」の「Width」「Height」ともに、単位は「1/1000mm」です。

この例では、「Width」(横幅)、「Height」(高さ)とも「5000」なので、画面上では5cmの大きさをもつ正方形になります。
(あくまでも理論上の大きさであり、正確にその大きさにならない場合もあるかもしれません)。

「Shape」を作って、各種属性を設定したら、シートの「DrawPage」で「DrawPage」を取得し、「Add」メソッドで「Shape」を追加します。

■「線」「塗り潰し」の設定

「Shape」の配置と、「位置」「大きさ」の設定ができるようになったので、続いて、「線」の設定や「塗り潰し」(背景)の設定を変えてみましょう。

「線」の設定を変えるには、「Shape」がもっている**「com.sun.star.drawing.LineProperties」**サービスのプロパティを変更します。

この「LineProperties」には、

LineStyle	…スタイル
LineDash	…スタイルでダッシュを指定したときの設定
LineColor	…色
LineWidth	…太さ

といったプロパティがあり、マクロからは「Shape」のプロパティとして変更可能です。

「LineStyle」には、「com.sun.star.drawing.LineStyle」で定義されている、以下の定数を指定します。

NONE	………線なし
SOLID	………実線
DASH	………ダッシュ

「DASH」のときには、「LineDash」に「com.sun.star.drawing.LineDash」構造体を設定することで、属性の設定ができます。

「LineColor」には、「16進数6桁」または「RGB( )関数で得られる色の値」を、「LineWidth」には、「線の太さ」を「1/100mm」単位で指定します。

*

塗り潰しの設定を変えるには、「Shape」がもっている「**com.sun.star.drawing.FillProperties**」サービスのプロパティを変更します。

この「FillProperties」には、

FillStyle	……スタイル
FillColor	……色

などのプロパティがあります。

「FillStyle」には「com.sun.star.drawing.FillStyle」で定義されている定数、

NONE	……………塗り潰しなし
SOLID	……………塗り潰し
GRADIENT	………グラデーション
HATCH	…………ハッチ
BITMAP	…………画像

のいずれかを指定します。

*

試しに、「2mmの太さで赤い線」「内部を青く塗り潰した正方形」を描いてみましょう。

```
Sub Main

    Dim objShape As Object
    Dim ptShape As New com.sun.star.awt.Point
    Dim sShape As New com.sun.star.awt.Size

    objShape = ThisComponent.createInstance("com.sun.star.
drawing.RectangleShape")

    ptShape.X = 6000
    ptShape.Y = 0

    sShape.Width = 5000
    sShape.Height = 5000

    objShape.setPosition(ptShape)
    objShape.setSize(sShape)

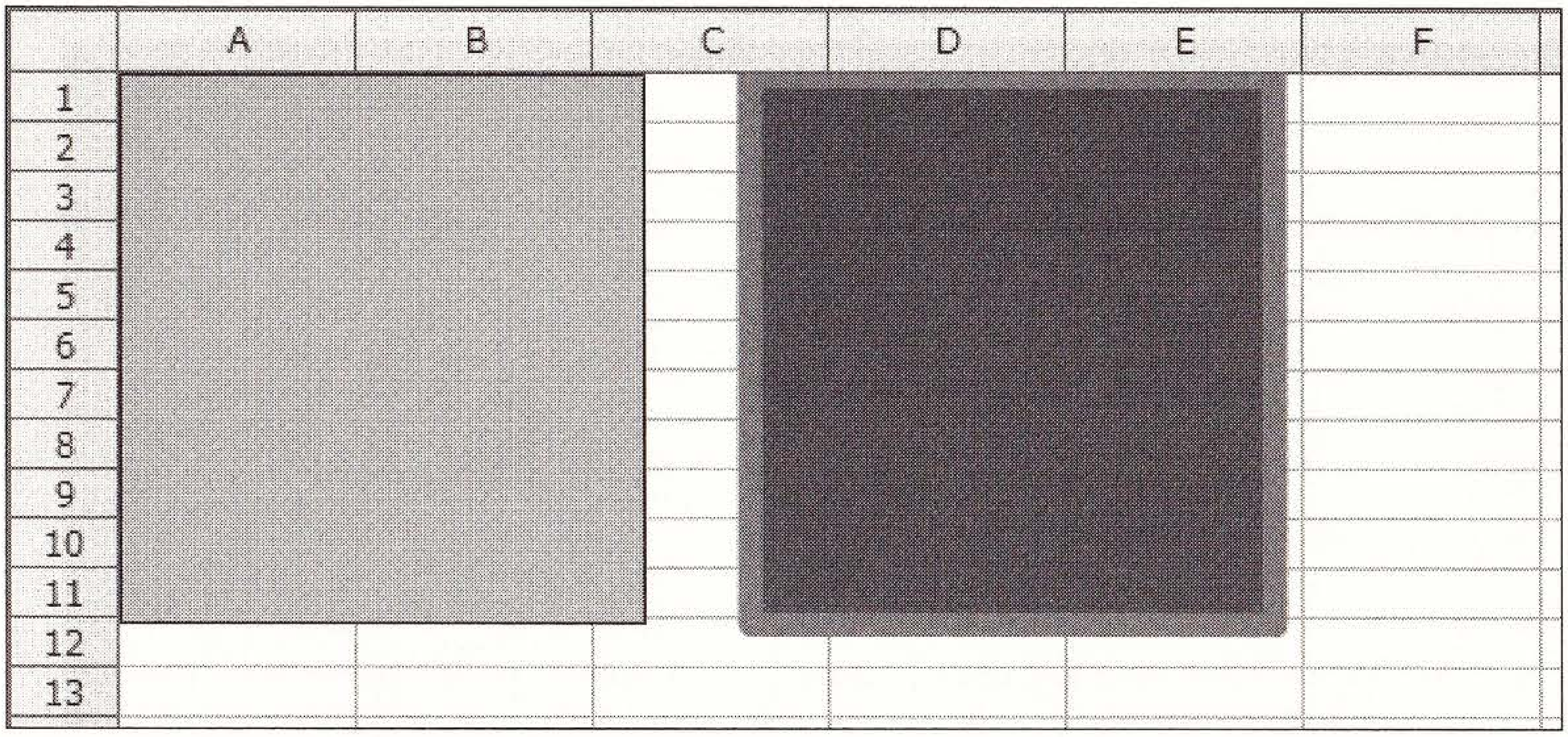
    objShape.LineColor = RGB(255, 0, 0)
    objShape.LineWidth = 200
```

```
    objShape.FillStyle = com.sun.star.drawing.FillStyle.SOLID
    objShape.FillColor = RGB(0, 0, 255)

    ThisComponent.Sheets(0).DrawPage.Add(objShape)

End Sub
```

実行すると、先のマクロで追加された正方形の右横に、新たに正方形が追加されます。



「線」「塗り潰し」が設定された正方形が配置される

■配置した図形を削除する

マクロから「Shape」の追加を試していると、マクロで描いた図形がたまってきて鬱陶(うっとう)しいので、「DrawPage」の「Shape」をすべて削除してみましょう。

「DrawPage」の「Shape」を削除するには、**「remove( )」**メソッドを使います。

引数には、削除する「Shape」を渡すので、「DrawPage」にある「Shape」を「DrawPage」の**「getByIndex( )」**メソッドで取得しましょう。

「getByIndex」は、「DrawPage」内の「Shape」を「インデックス」(「0」からはじまる番号)で指定しますが、とりあえず、「今DrawPageにあるShapeを一つ取得する」のなら、「getByIndex(0)」で取得できます。

そして、こうして取得した「DrawPageにあるインデックス0のShape」を、「DrawPage」の「getCount( )」で取得できる「DrawPage」の「Shape数」が「0」になるまで繰り返せば、「DrawPageにあるShapeをすべて削除」できるはずです。

＊

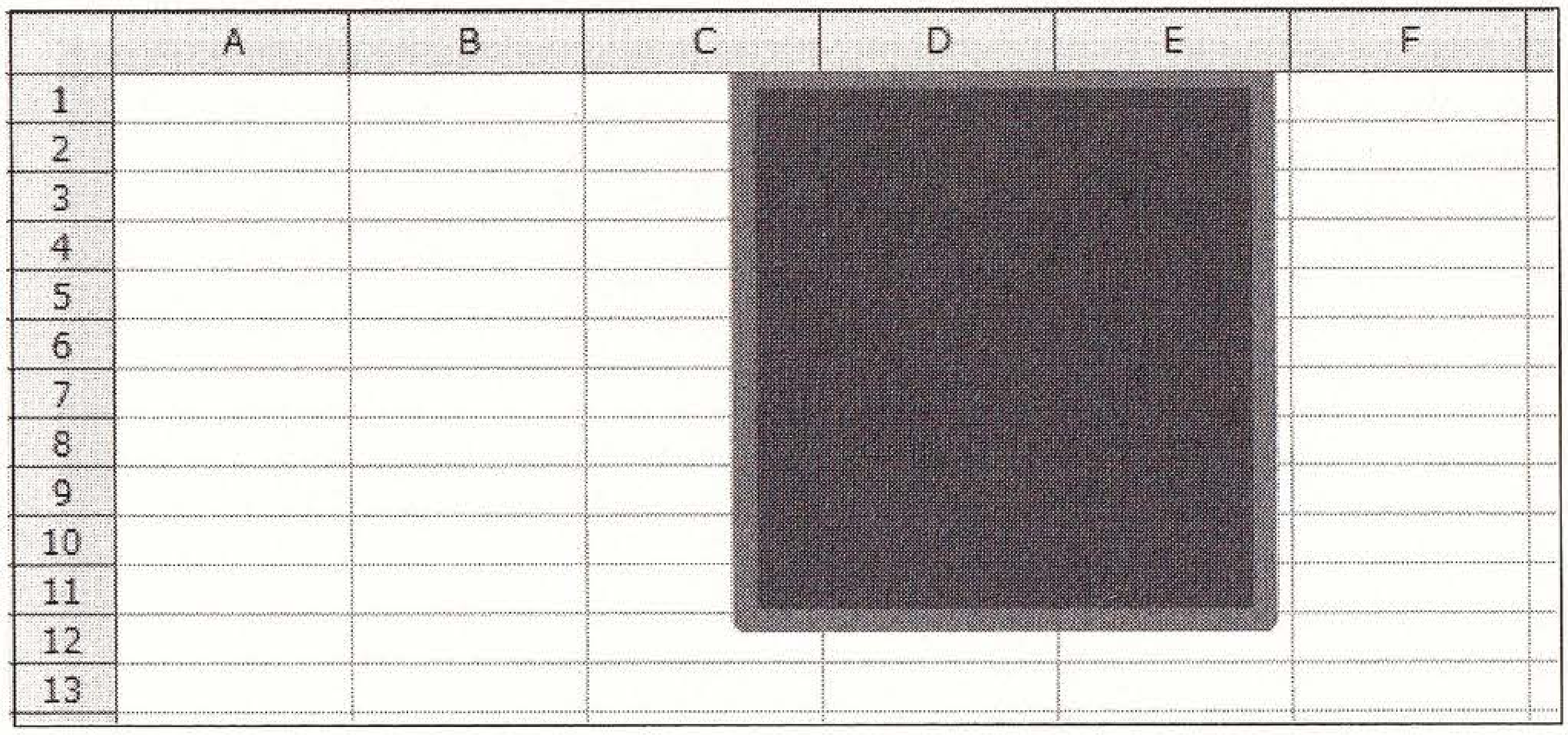
試しに、これまでのマクロの先頭に以下の処理を追加して実行してみてください。

```
Dim page As Object

page = ThisComponent.Sheets(0).DrawPage

While page.getCount() > 0
    page.remove(page.getByIndex(0))
Wend
```

これまでに実行したマクロなどで作った「Shape」が削除され、新しく作った「Shape」だけが表示されるようになったはずです。



古い図形が削除された

＊

今回は、とりあえず「RectangleShape」(長方形)を試してみましたが、「Shape」には他にも、

EllipseShape	楕円
GraphicObjectShape	画像
LineShape	線
TextShape	文字列

など、さまざまなものがあります。

4-5 Excelの「VBAマクロ」を使う

「Calc」では、「Microsoft Office」の表計算ソフト「Excel」の表計算シートを読み込むことができます。

エクセルには、「Calc」の「OpenOffice Basicマクロ」と同じような機能をもつ「VBAマクロ」があり、最近の「Calc」では、ExcelのシートにあるVBAマクロも使えるようです。

『Excel上のVBAでアプリケーションを開発してきて、多数の「資産」を抱えている』『普段VBAでアプリケーションを開発している』という人にとっては、この「Calc」のVBA実行機能はなかなか興味深い機能でしょう。

■VBAマクロをCalc上で動作させてみる

簡単なVBAマクロを含むエクセルのシートを「Calc」で開いてみて、実際にどの程度動くのか見てみましょう。

まず、Excel上で以下のVBAマクロを含むシートを作ります。
(手元にあるのが「Excel2000」なので、それ以降のバージョンでは少し違う部分があるかもしれません)。

```
Sub test1()

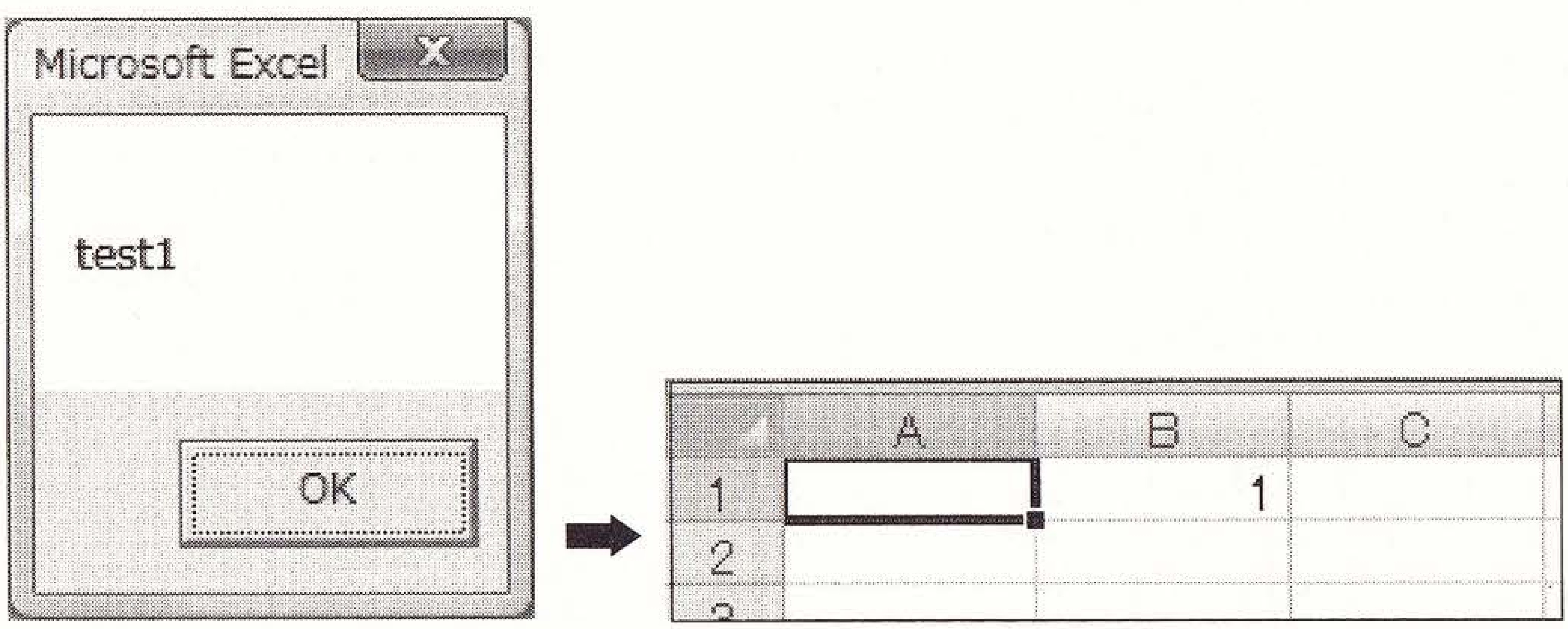
    MsgBox "test1"

    Cells(1, 2).Value = 1

End Sub
```

これは、メッセージ・ボックスで「test1」という文字列を表示した後、セル「B1」に「1」という値を設定するマクロです。

エクセル上で実行すると、「test1」というメッセージ・ボックスが出てきて、「B2」セルに「1」という値が入りました。



実行結果

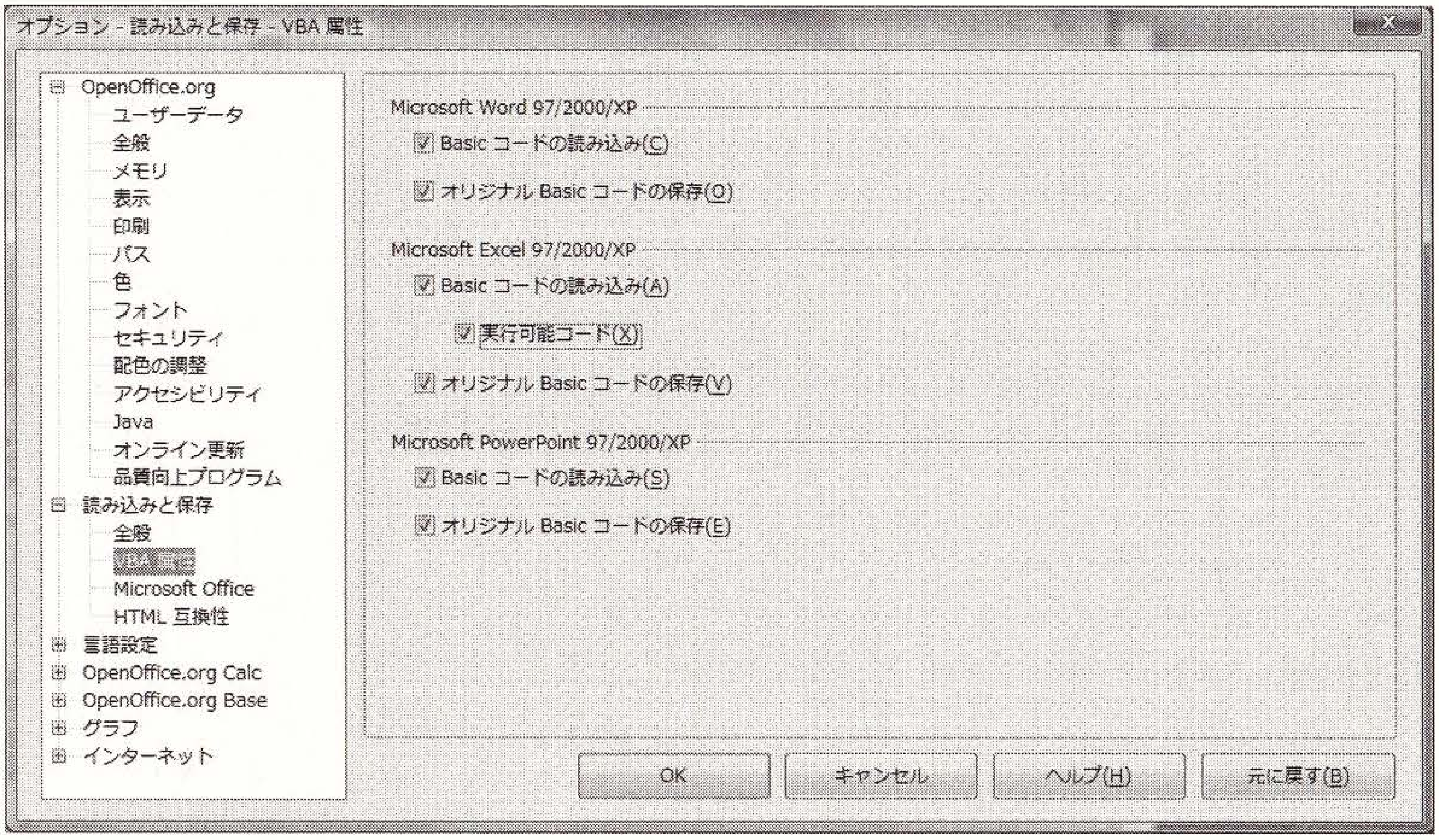
「Calc」上でこのマクロを正常に実行できれば、メッセージやセルへの値設定というVBAによる基本的な出力を、「Calc」でも同様にできることを意味します。

Excel上での動作を確認したら、適当な名前で保存して、「Calc」を起動してください。

＊

「Calc」を起動したら、先ほどのExcelファイルを読み込む前に、「VBAマクロ」を実行できるように設定を変更します。

ツール・メニューから「オプション」設定画面を立ち上げ、「読み込みと保存」にある「VBA属性」をクリックしてください。



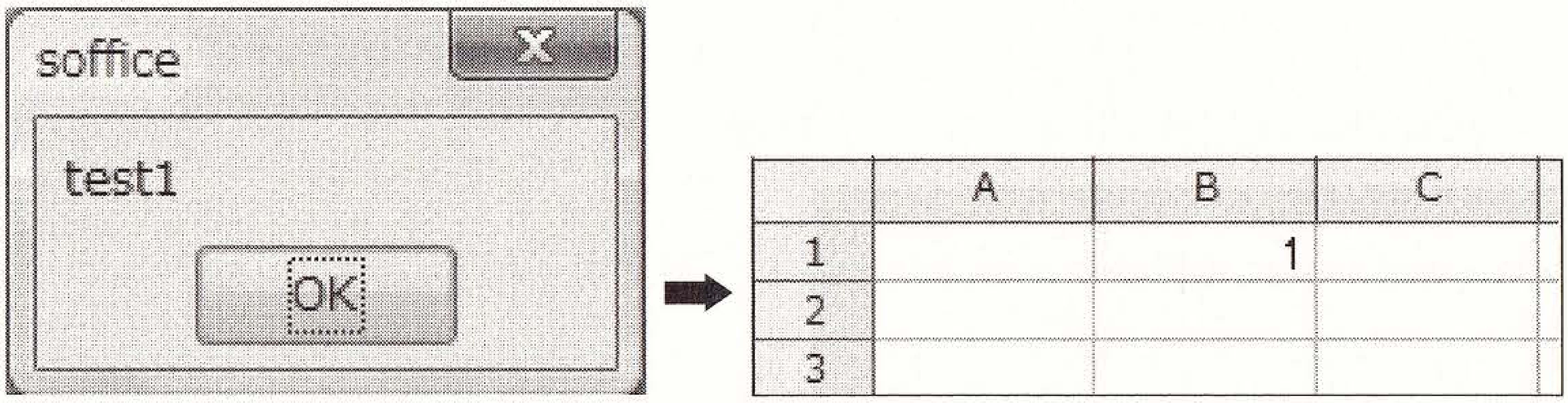
「オプション」設定画面

ここに、VBA読み込みに関するいくつかのオプションがあるので、これらをすべて選択状態にします。
(初期状態では、「実行可能コード」のチェックが外されていて、VBAマクロが実行できないようです)。

＊

設定を終えたら、「開く」メニューから先ほどのExcelファイルを開いてみましょう。

マクロに関する警告が出てくるので、マクロを「有効」にします。
続いて、「ツール」内のマクロ・メニューから「マクロを実行」を選ぶと、マクロの選択画面になるので、読み込んだシートの「test1」を実行してみてください。



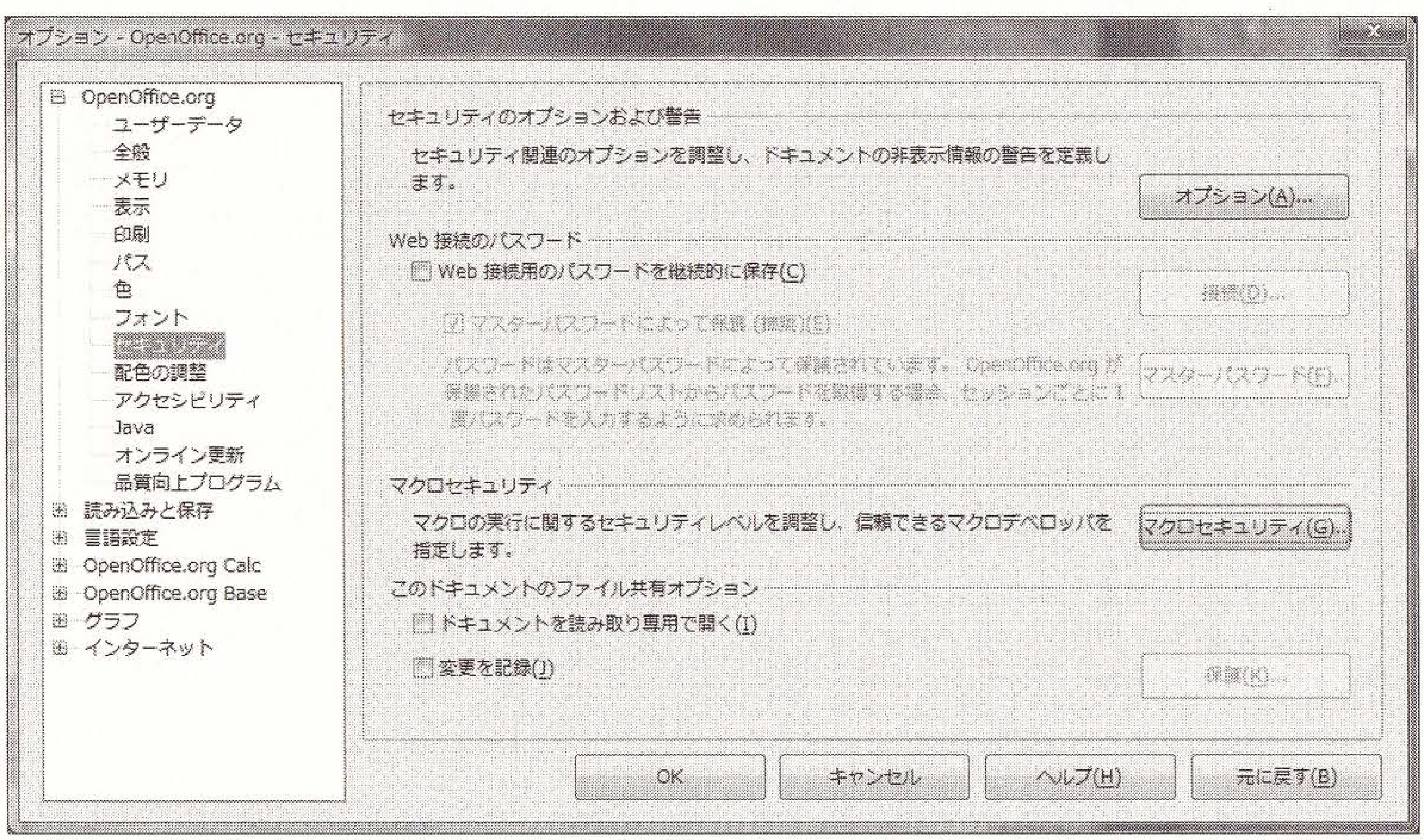
実行結果

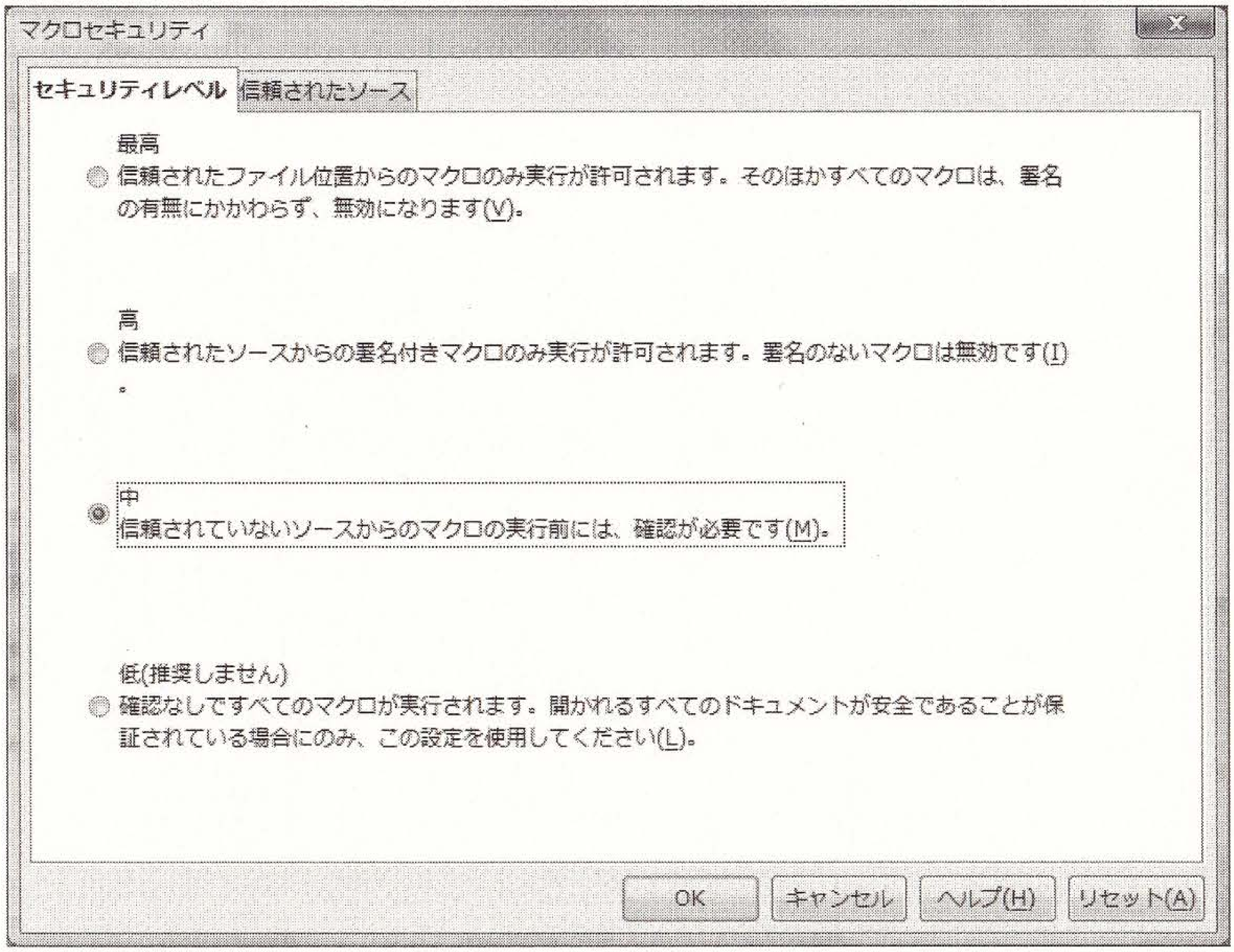
エクセルのときと同じく、メッセージ・ボックスが出てきて、「B1」セルに値が設定されました。

もし、実行されない場合は、最初に設定した「オプション」設定画面を呼び出し、「OpenOffice.org」内のセキュリティ欄で「マクロセ・キュリティ」を「中」などに変更してみましょう。

ただし、設定を変更すると、「悪意あるマクロ」を含むシートを開いた場合に、何らかの問題が生じる可能性があります(「Calc」上では、「VBAマクロ」からのファイルの読み書きも可能)。

特にセキュリティが重視される環境では、マクロの実行を許可すべきか、慎重に検討してください。





「マクロセ・キュリティ」の設定画面

■CSVファイルを読み込んでみる

次に、「CSVファイル」の読み込みを試してみます。

Excelのシートに以下のようなマクロを追加します(CSVのファイル・パスは、環境に合わせて書き換えてください)。

CSVファイル「d:¥test.csv」を読み込み、 その内容を左上端からセルに設定する。

適当なCSVファイルを作ったら、実際にマクロを実行して、セルに値が設定されることを確かめてください。

	A	B	C
1	1	2	3
2	A	B	C
3	あ	い	う
4			

実行結果

マクロの構文や関数の扱い方は、「Calc」の「OpenOffice.org BASICマクロ」とよく似ているので、「VBA」を使ったことがない人でも、一読すれば、なんとなく意味が読み取れるのではないでしょうか。

```
Sub test2()

    Dim fh As Integer
    Dim csvLine As String
    Dim csvItems As Variant
    Dim rowIndex As Integer
    Dim colIndex As Integer

    fh = FreeFile

    Open "d:¥test.csv" For Input As #fh

    rowIndex = 0

    Do While Not EOF(fh)

        Line Input #fh, csvLine

        csvItems = Split(csvLine, ",")
```

```
        For colIndex = 0 To UBound(csvItems)
            Cells(rowIndex + 1, colIndex + 1).Value = csvItems
(colIndex)
        Next

        rowIndex = rowIndex + 1

    Loop

    Close #fh

End Sub
```

適当な名前で保存したら、「Calc」で開いてマクロを実行してみましょう。

Excelのときと同様に、問題なく処理されたと思います。「配列」や「Variant変数」周りの処理も大丈夫のようです。

	A	B	C
1	1	2	3
2	A	B	C
3	あ	い	う
4			

CSVファイルもExcelと同じように処理される

*

以上のことから、「Calc」では、

- **VBAと同様に、ファイルを開ける。**
- **VBAと同様に、CSVファイルの各行各項目を(配列要素として)取り出せる。**
- **VBAと同様に、各セルに値を設定できる。**
- **VBAと同様に、任意の文字列をメッセージ・ダイアログに表示できる。**

ことが分かりました。

つまり、「Calc」でもVBAを含むExcelシートを使って、「データ・ファイルをセルに読み込んでその内容に対し、OpenOffice.org BASICで記述した処理を実行できる」わけです。

これなら、一般的な統計処理を行なうExcelシートの基本機能については、大きな問題なく使える(「Excel」と「Calc」の両方で使える「VBAマクロ」を含む表計算シートが作れる)可能性が高い、と言えるのではないでしょうか。

4-6　データベース「Base」

「OOo」には、「Base」というデータベース・アプリケーションが含まれています。

「Base」は、「データベース」(otdファイル)の中に複数の「テーブル」(表)を格納するデータベース・システムで、「データベース」の実体はデータベースごとに作られる「otdファイル」そのものです。

ここでは、この「Base」のデータベースをマクロから扱ってみたいと思います。

「otdファイル」に対しては、「OOo」のシステムを通して「SQL」(データベースを扱うためのプログラミング言語)でデータの処理を行なうことができます。

「OpenOffice.org Basicマクロ」からは、「OOo」がもっているデータベースへの接続を取得し、「SQL」でアクセスできるので、テスト用に作ったデータベースのデータを「SQL」で読み込んでみましょう。

■「Base」でテスト用データベースを作る

まず、「Base」を起動し、テスト用のデータベースを作ります。

「Base」を起動すると、編集するデータベースの選択画面になるので、「新規データベースの作成」で、「addressBook」など、適当な名前のデータベースを作ってください。

この際、「OpenOffice.orgにデータベースを登録しますか」の項目が「はい」になっているのを確認します。

＊

データベース・ファイルを保存し、「データベースの編集」画面になったら、テスト用テーブル「ADDRESSBOOK」を作りましょう。

「ADDRESSBOOK」の各データには、

no(番号) ……………………整数型、主キー
name(名前) ………………文字列型
mail(メールアドレス)………文字列型

の3つの項目を格納するようにします。

まず、「ツール」メニューの「SQL」で、「SQLステートメントの実行」ダイアログを出してください。

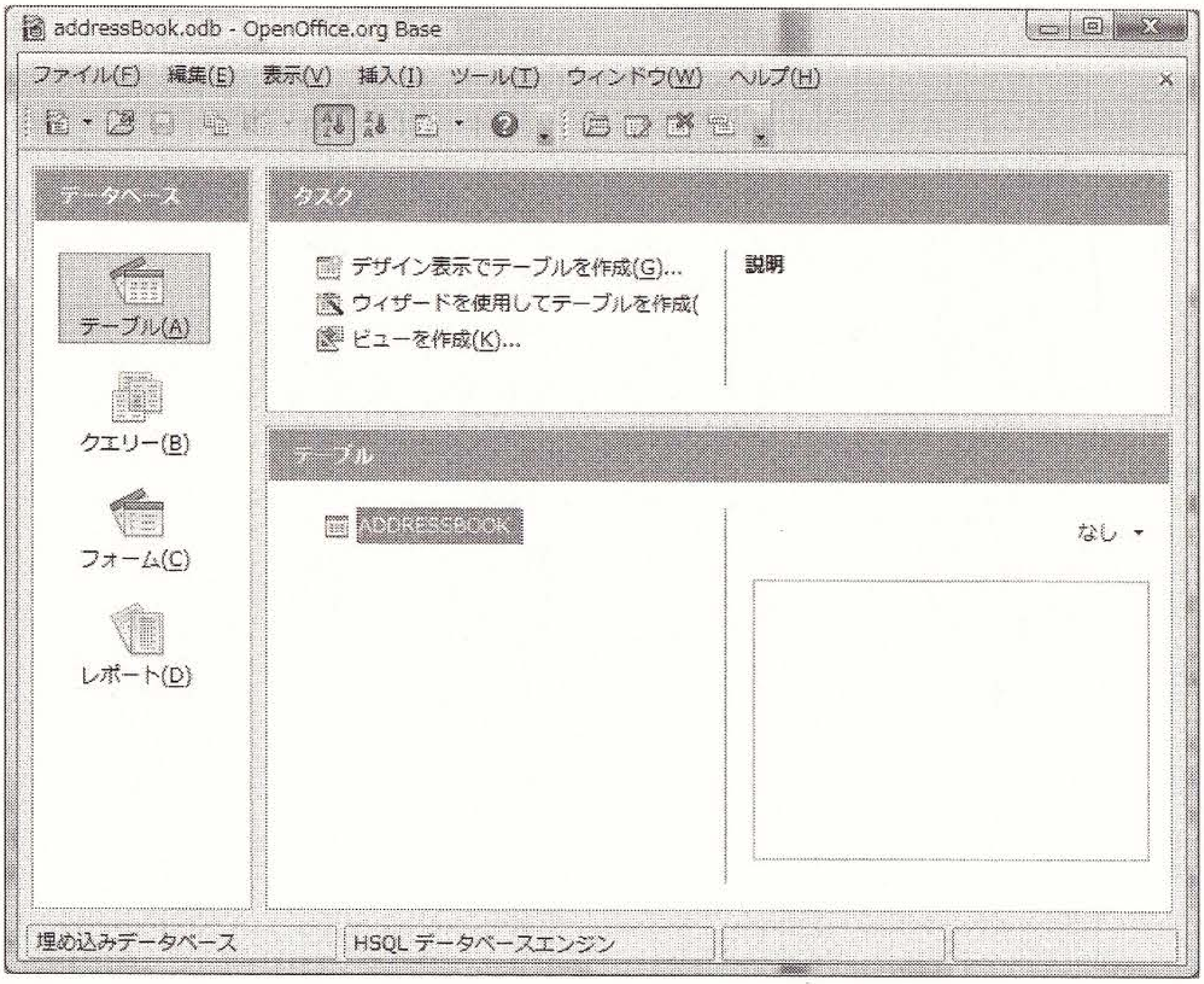
「実行コマンド」欄に、以下のようなテーブルを作る「SQL」を入力し、「実行」すると、テーブルが作られます。

```
CREATE TABLE ADDRESSBOOK(no INT GENERATED BY DEFAULT AS
IDENTITY (START WITH 1) NOT NULL PRIMARY KEY, name VARCHAR
(32) NOT NULL, mail VARCHAR(64))
```



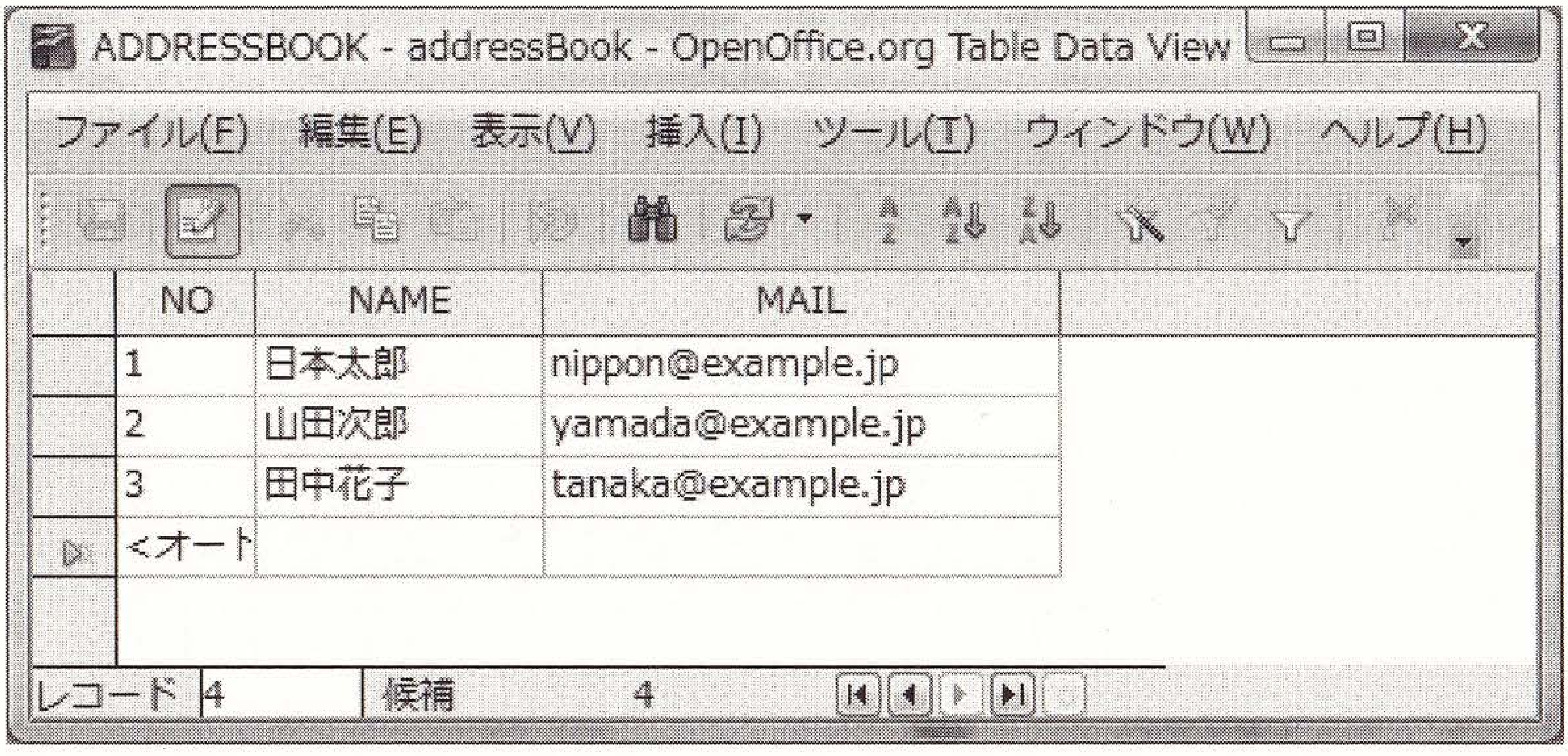
「SQLステートメントの実行」でテーブルを作成

「SQL」を実行したら、ダイアログを閉じて、「表示」メニューから「テーブルの更新」を行ないましょう。これで、「テーブル」欄に今作った「ADDRESSBOOK」テーブルのアイコンが見える状態になったはずです。



「ADDRESSBOOK」テーブルが追加された

アイコンをクリックして、何件か適当にデータを入力してください。



「ADDRESSBOOK」テーブルにデータを登録

■データベースの登録名を確認する

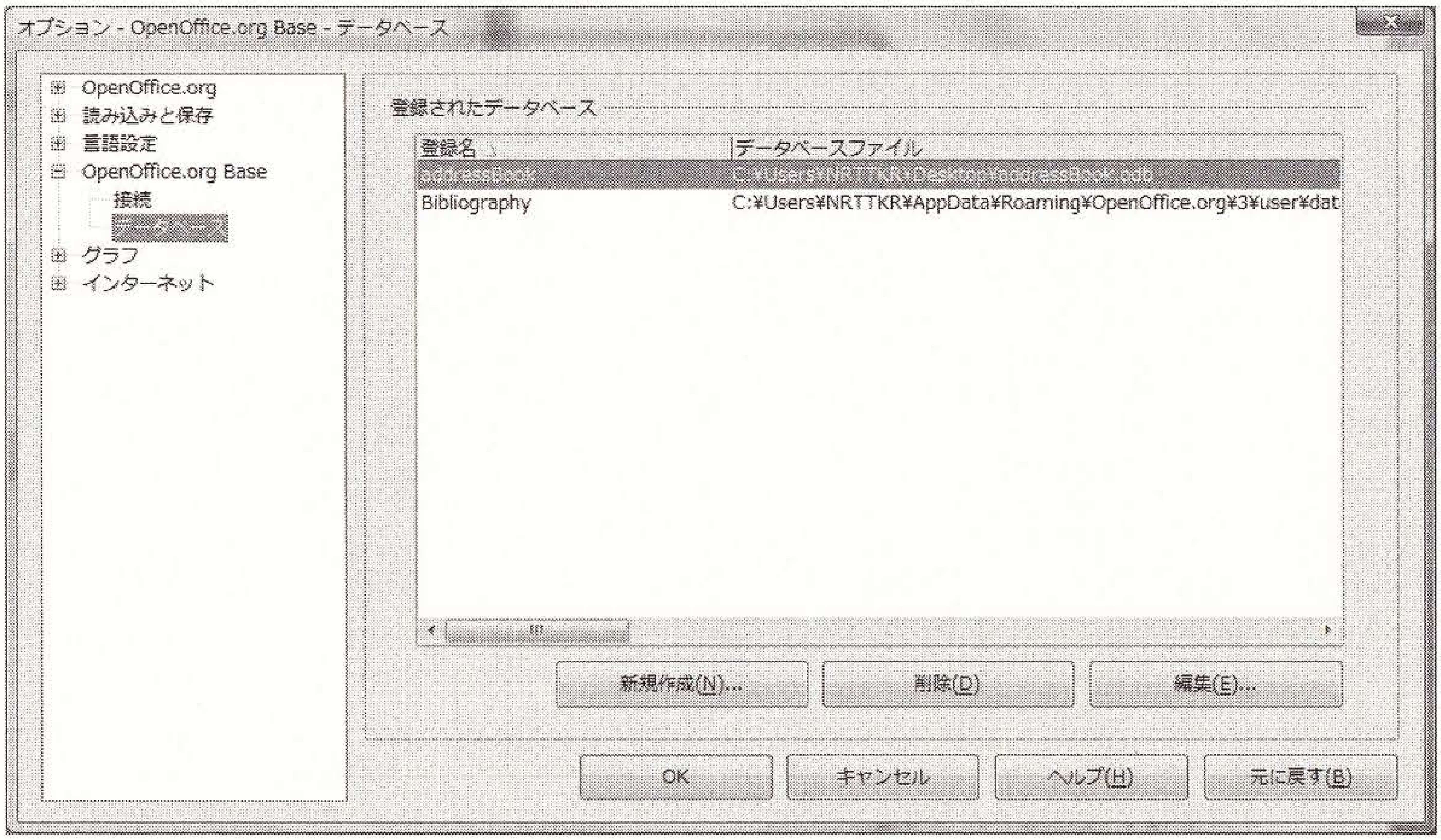
何件かデータを追加し、保存したら、今作っているデータベースが実際に「OOo」のシステム上でデータベースとして認識されているか、確認しましょう。

「ツール」内の「オプション」メニューで、「OOo」の「設定」画面を呼び出してください。

次に、ツリー・ビューの「OpenOffice.org Base」をクリックすると「接続」「データベース」という項目が出てくるので、「データベース」を選びます。

すると、現在登録されているデータベースが「登録されたデータベース」として一覧表示されるので、そこに先ほど作ったデータベースがあることを確認しましょう。

この「登録名」が、マクロからデータベースを使うときに必要になります。



データベースの登録名を確認

「登録名」を確認したら、一度保存し、「Base」を終了してください。

■マクロからデータベースにアクセスする

改めて「Base」を再起動したら、いつものように「ツール」メニューの「マクロ」から、マクロの編集画面に入ってください。

さっそく、先ほど作ったデータにマクロからアクセスしてみましょう。

マクロからデータベースを利用する処理は、

[1]「com.sun.star.sdb.DatabaseContext」サービスで「接続名」を指定して「データ・ソース」を取得。

[2]「データ・ソース」から「Connection」を取得。

[3]「Connection」から「Statement」を作る。

[4]「Statement」の「executeQuery( )」で「SQL」を実行し、抽出結果の「ResultSet」を取得。

[5]「ResultSet」から「データ」を取得。

という流れで行ないます。

たとえば、接続名「addressBook」にあるテーブル「ADDRESSBOOK」のデータをすべて含む「ResultSet」を得て、その最初のデータの2番目の項目(「ADDRESSBOOK」テーブルの「name」)を表示するなら、

```
Sub Main

    Dim context As Object
    Dim src As Object
    Dim cnn As Object
    Dim stm As Object
    Dim rst As Object

    context = createUnoService("com.sun.star.sdb.DatabaseContext")

    src = context.getByName("addressBook")

    cnn = src.getConnection("", "")

    stm = cnn.createStatement()

    rst = stm.executeQuery("SELECT * FROM ADDRESSBOOK")

    If Not IsNull(rst) Then

        rst.next

        MsgBox rst.getString(2)

    End If

    rst.close()

    cnn.close()
```

```
    cnn.dispose()

End Sub
```

のようにします。

「executeQuery( )」で得られる「ResultSet」は、「SQL」で抽出されたデータを保持する「コレクション」で、複数のデータを格納することができます。



「name」項目が表示される

＊

データにアクセスするには、

next …………次のデータに移動	
previous ……前のデータに移動	

といったメソッドで、「現在位置」を変えながら参照していきます。

ただし、最初の状態では、「ResultSet」の現在位置が「最初のデータの前」(無効な位置)になっている点に注意が必要です。

「ResultSet」に含まれる最初のデータを参照するには、「**next( )**」で現在位置を1つ進める必要があります。

現在位置を有効なデータのある位置に進めたら、「**getString( )**」など、行に含まれるデータを取得するメソッドでデータを取得できます。

「getString( )」は、引数にデータの位置を「1」からはじまる数字で指定すると、その位置にあるデータを文字列として返す関数です。

「next()」は、実際に移動できたら「TURE」、できなければ「FALSE」を返すので、「next( )」が真である間、「next( )」で次のデータを処理するループを書けば、「ResultSet」の全データを処理できます。

たとえば、「ADDRESSBOOK」の「データ」を、すべて「Calc」のシートに書き出すには、以下のようにします。

「Calc」を起動し、以下のマクロを実行してみてください。

```
Sub Main

    Dim context As Object
    Dim src As Object
    Dim cnn As Object
    Dim stm As Object
    Dim rst As Object

    context = createUnoService("com.sun.star.sdb.DatabaseContext")

    src = context.getByName("addressBook")

    cnn = src.getConnection("", "")

    stm = cnn.createStatement()

    rst = stm.executeQuery("SELECT * FROM ADDRESSBOOK")

    rowIndex = 1

    If Not IsNull(rst) Then
        While rst.next

            For i = 1 to 3
```

```
                ThisComponent.Sheets(0).getCellByPosition(i,
rowIndex).String = rst.getString(i)
            Next

            rowIndex = rowIndex + 1

        Wend
    End If

    rst.Close()

    cnn.Close()
    cnn.Dispose()

End Sub
```

	A	B	C	D	E
1					
2		1	日本太郎	nippon@example.jp	
3		2	山田次郎	yamada@example.jp	
4		3	田中花子	tanaka@example.jp	
5					

データベースの内容がCalcのシートに反映される

第5章

CSVファイルの「整形」「自動処理」

「CSVファイル」を「Calc」に読み込ませる際に、目的の項目のみを読み込む、一定の条件を満たすものを抽出して表示する、といった整理方法をマクロで実現してみましょう。

5-1 CSVファイルを特定の条件で処理する

銀行や証券会社のオンライン・サービスでは、取引履歴を「CSVファイル」でダウンロードできる場合があります。ただ、「CSVファイル」といっても形式はまちまちです。

不要なデータもあったりして、「CSV」のまま「Calc」に読み込ませて使うと、ちょっと扱いにくい場合もあります。また、特定の場所に保存してある「CSVファイル」をまとめて処理したい、ということもあるかもしれません。

ここでは、そうした「CSVファイル」の「整形」や「自動処理」をマクロで行なうために、「項目の抽出」や「日付によるフィルタ」など、基本的な流れを試してみましょう。

■項目を抽出する

今回扱う「CSV」は、以下の形式のものとします。

```
年,月,日,種別(1-入金/2-出金),入金額,出金額
```

この形式の「CSVファイル」を読み込んで、1件1行で「年月日、入金額、出金額」の順にセルに出力することにしましょう。

出力先は、マクロで自動作成した「新規シート」とします。

*

まず、出力するシートを作ります。

シートの名前は「testSheet51」とし、すでに同名のシートがあれば、一度削除してから新規作成するようにしました。

```
Dim doc As Object
Dim sheets As Object
Dim sheetName As String
```

```
doc = ThisComponent

sheetName = "testSheet51"

sheets = doc.Sheets

If sheets.hasByName(sheetName) Then
    sheets.removeByName(sheetName)
End If

sheets.insertNewByName(sheetName, 0)
```

*

続いて、「CSVファイル」を読み込み、1行ずつセルに転記してみましょう。

転記するのは、「年月日」「入金」「出金」とし、「年月日」は「年/月/日」と、ひとつにまとめます。

まず、以下のような「CSVファイル」を作って、適当な場所に保存してください。

```
2009,7,1,1,10000,0
2009,7,2,2,0,10000
2009,7,3,2,0,20000
2009,7,4,1,0,30000
```

次に、以下のマクロを作って、「**csvPath**」に「CSVファイル」の場所を指定します。

```
Sub Main

  Dim doc As Object
  Dim sheets As Object
  Dim sheetName As String

  Dim fineNo As Integer
  Dim csvLine As String
  Dim csvItems() As String
  Dim csvPath As String
  Dim rowIndex As Long
  Dim date As String

  csvPath = "file:///e:/test.csv"

  doc = ThisComponent

  ' シート名設定
  sheetName = "testSheet51"

  sheets = doc.Sheets

  ' 既に同名のシートがあれば削除
  If sheets.hasByName(sheetName) Then
        sheets.removeByName(sheetName)
  End If

  ' シートを追加
  sheets.insertNewByName(sheetName, 0)

  fileNo = Freefile

  ' CSVファイルを開く
  Open csvPath For InPut As #fileNo

  ' 書き込むセルの行位置を設定
```

```
  rowPos = 0

  ' CSV各行を処理
  While not eof(fileNo)

        ' 一行読み込み
        Line Input #fileNo, csvLine

        ' 各項目を分離
        csvItems = Split(csvLine, ",")

        ' 日付の文字列を作成
        date = csvItems(0) & "/" & csvItems(1) & "/" & csvItems(2)

        sheets(0).getCellByPosition(0, rowPos).String = date
        sheets(0).getCellByPosition(1, rowPos).String = csvItems(4)
        sheets(0).getCellByPosition(2, rowPos).String = csvItems(5)

        ' 行位置を更新
        rowPos = rowPos + 1

  Wend

  Close #fileNo

End Sub
```

マクロでは、まず、「CSVファイル」を開いて、1行ずつ読み出すループに入ります。

ループ内では、

1番目の項目/2番目の項目/3番目の項目

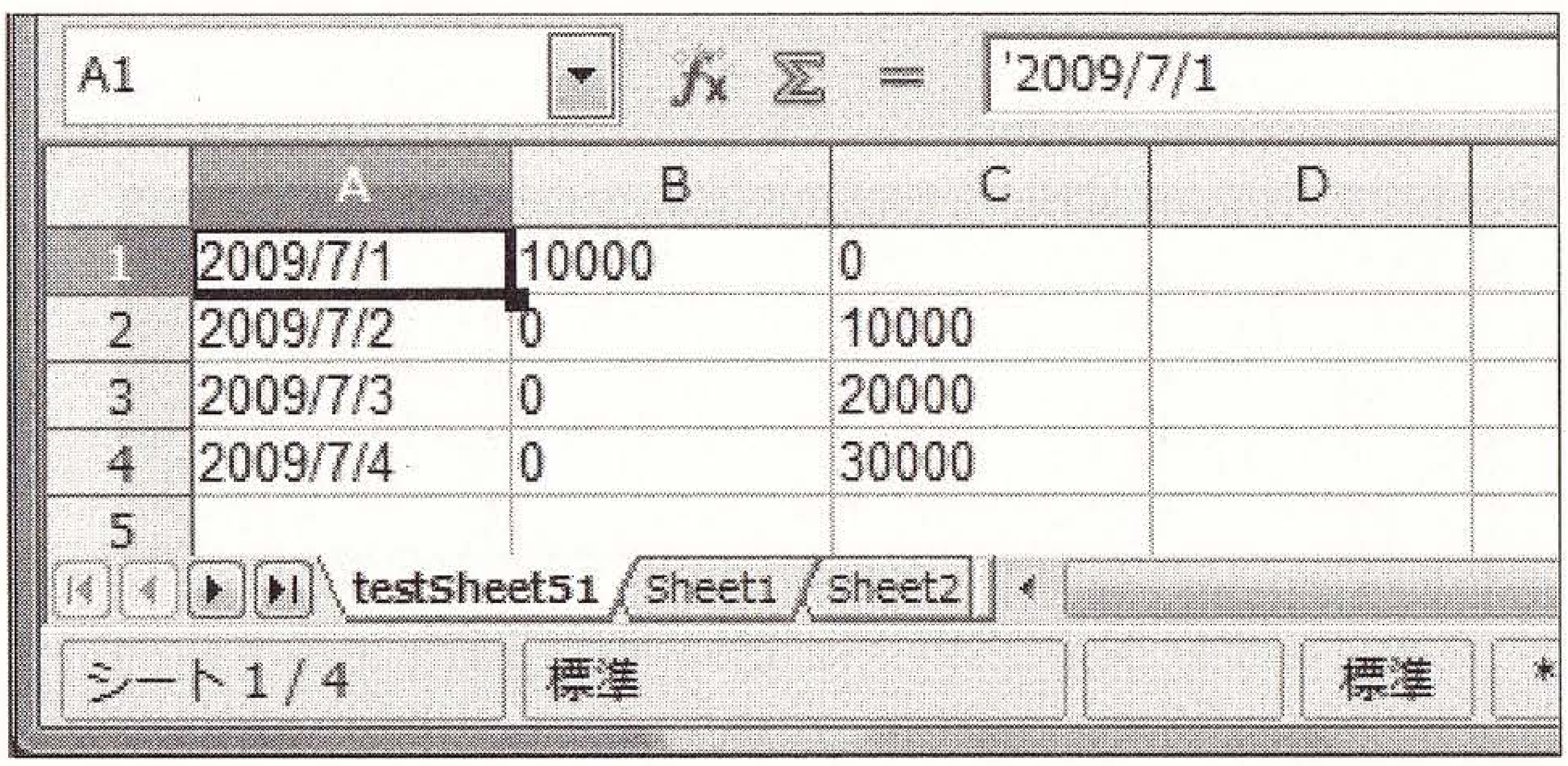
という形で作った「日付文字列」を、各行の左端セル(インデックス「0」)に入れます。

続いて、「入金」(CSVの5番目の項目)を、左から2番目(インデックス「1」)のセルに、「出金」を左から3番目のセル(インデックス「2」)のセルに入れ、現在の行位置を更新します。

これを、「CSVファイル」の最後まで繰り返してみました。

＊

マクロを実行すると、「testSheet51」という新規シートが作られているので、そのシートを開いて実行結果を確認してください。



「CSVファイル」を実行結果

■特定の条件上で、項目を抽出する

次に、「一定期間のデータだけを表示する」ようにしてみましょう。

「日付型のデータ形式にして大小を比較」という手もありますが、今回は簡単に、「4桁の年、2桁の月と日、合計8桁の数値」で比較することにします。

つまり、年月日を、

年×10000+月×100+日

の数値として扱うわけです。

あらかじめ、変数「startDate」「endDate」に「開始日」と「終了日」を入れておいて、CSVの「年月日」がその間にあれば、データを表示するようにします。

CSVの「年月日」を「数値」として計算する際は、「Long型」数値に変換する「**CLng( )**」関数を使うようにしました。

```
Sub Main

  Dim doc As Object
  Dim sheets As Object
  Dim sheetName As String

  Dim fineNo As Integer
  Dim csvLine As String
  Dim csvItems() As String
  Dim csvPath As String
  Dim rowIndex As Long
  Dim date As String

  Dim startDate As Long
  Dim endDate As Long
  Dim itemDate As Long

  csvPath = "file:///e:/test.csv"

  ' 期間設定
  startDate = 20090702
  endDate = 20090704

  doc = ThisComponent

  ' シート名設定
  sheetName = "testSheet51"

  sheets = doc.Sheets
```

```
' 既に同名のシートがあれば削除
If sheets.hasByName(sheetName) Then
    sheets.removeByName(sheetName)
End If

' シートを追加
sheets.insertNewByName(sheetName, 0)

fileNo = Freefile

' CSVファイルを開く
Open csvPath For InPut As #fileNo

' 書き込むセルの行位置を設定
rowPos = 0

' CSV各行を処理
While not eof(fileNo)

    ' 一行読み込み
    Line Input #fileNo, csvLine

    ' 各項目を分離
    csvItems = Split(csvLine, ",")

    ' 日付の文字列を作成
    date = csvItems(0) & "/" & csvItems(1) & "/" & csvItems(2)

    ' 日付を数値化
    itemDate = CLng(csvItems(0)) * 10000 + CLng(csvItems
(1)) * 100 + CLng(csvItems(2))
```

```
      ' 日付が期間内ならシートに追加
      If itemDate >= startDate And itemDate <= endDate Then

         sheets(0).getCellByPosition(0, rowPos).String = date
         sheets(0).getCellByPosition(1, rowPos).String = csvItems(4)
         sheets(0).getCellByPosition(2, rowPos).String = csvItems(5)

         ' 行位置を更新
         rowPos = rowPos + 1

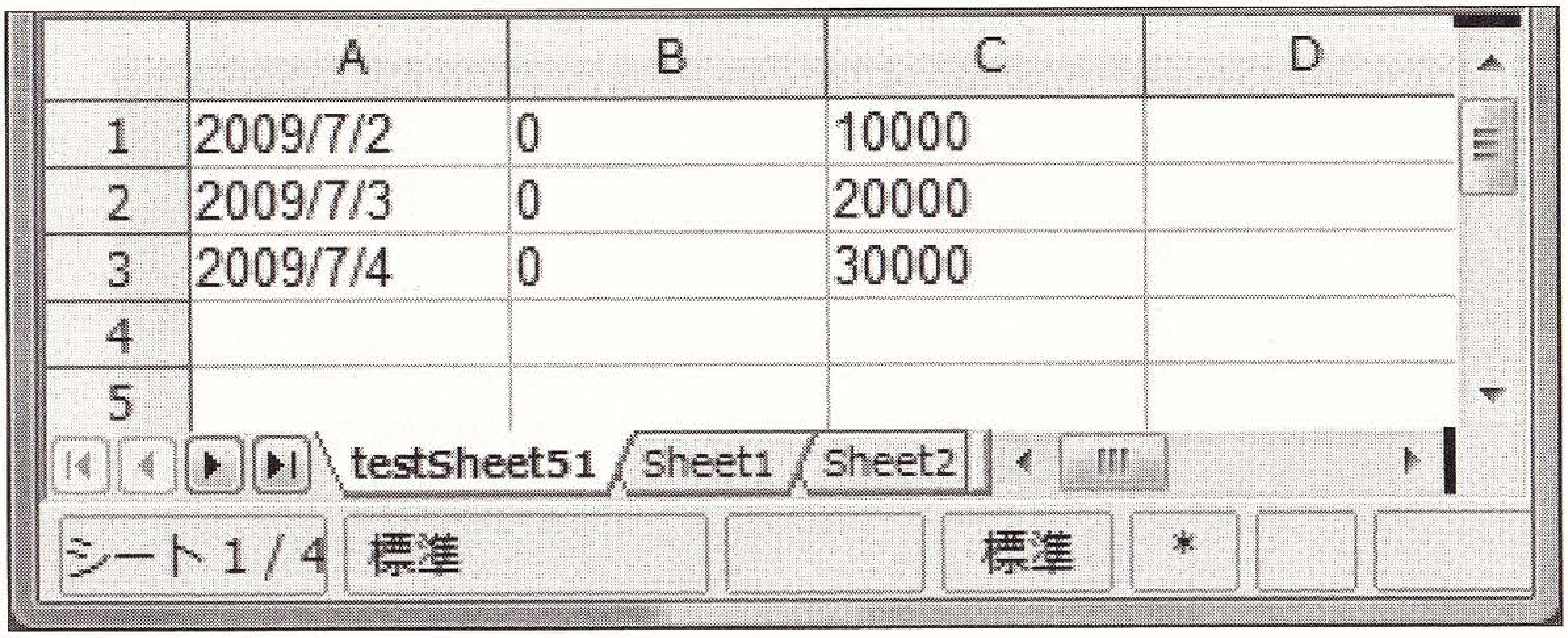
      End If

   Wend

   Close #fileNo

End Sub
```

マクロを実行してみると、「startDate」(20090702)と「endDate」(20090704)で指定した期間のデータだけが表示されたと思います。



	A	B	C	D
1	2009/7/2	0	10000	
2	2009/7/3	0	20000	
3	2009/7/4	0	30000	
4				
5				

実行結果

*

以上、「CSVファイル」を読み込んで、「特定の条件で抽出したデータ」の「特定の項目」をセルに書き出す処理を試してみました。

「CSVファイル」の形式は、データの提供者によってさまざまなので、マクロなどで一度分かりやすい形に整理してから、処理する必要が出てくることもあるでしょう。

*

なお、今回は省きましたが、実際に「CSVファイル」を処理するマクロを作る際には、「ファイル読み込みの失敗」や「CSVデータの異常」など、エラー対策にも注意してください。

MEMO

付　録

【付録1】「OpenOffice.org」のインストール
【付録2】「OpenOffice.org」の派生ソフト「Go-OO」
【付録3】困ったときのために
【付録4】「OpenOffice.org Basic」予約語一覧

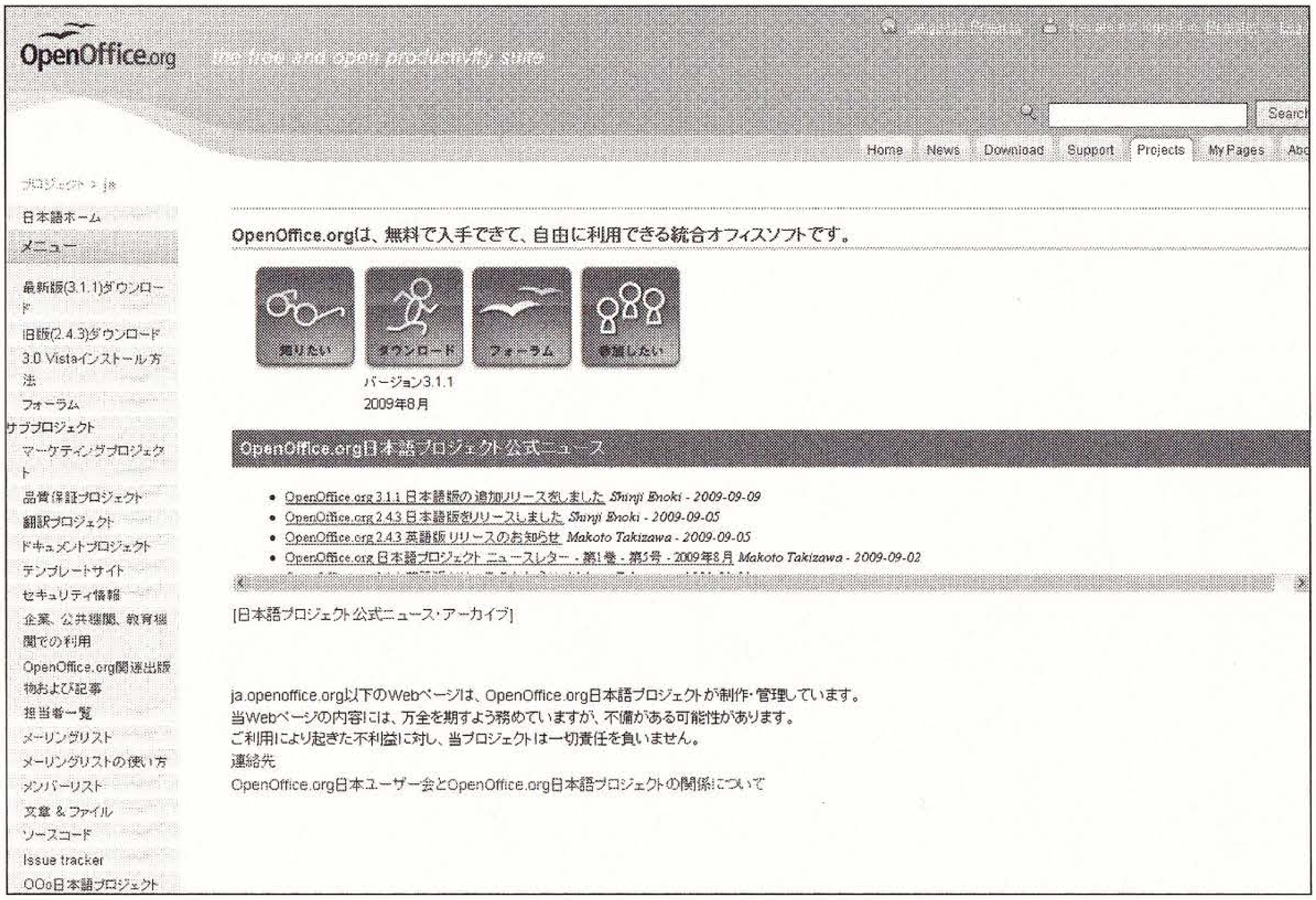
付録1 「OpenOffice.org」のインストール

ここでは、「OOo」のインストール方法を説明します。

■「OpenOffice.org」のインストール

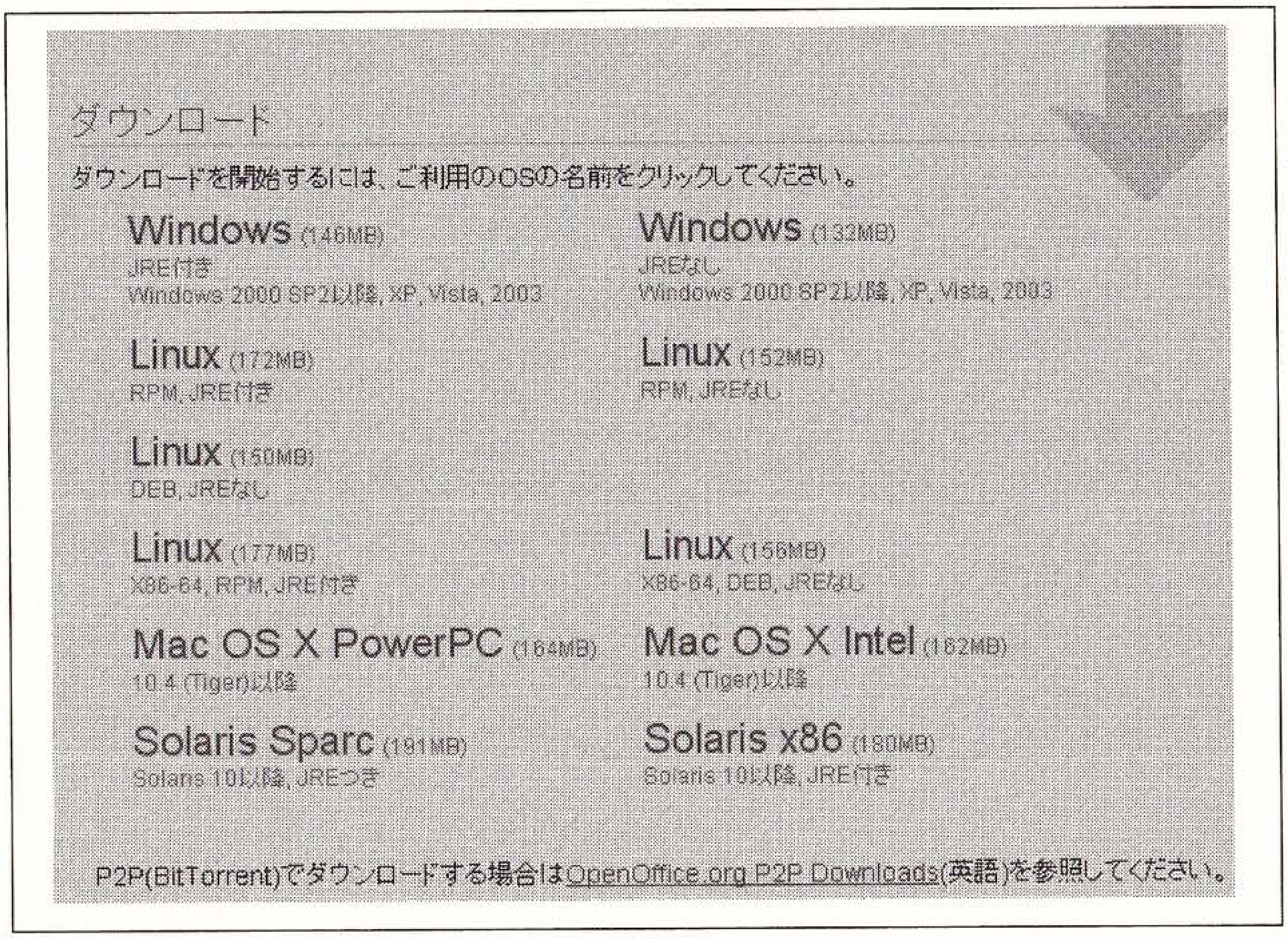
【手順】

[1]「OpenOffice.org日本語プロジェクト」のWebサイトに行き、トップページの「ダウンロード」をクリックしてください。



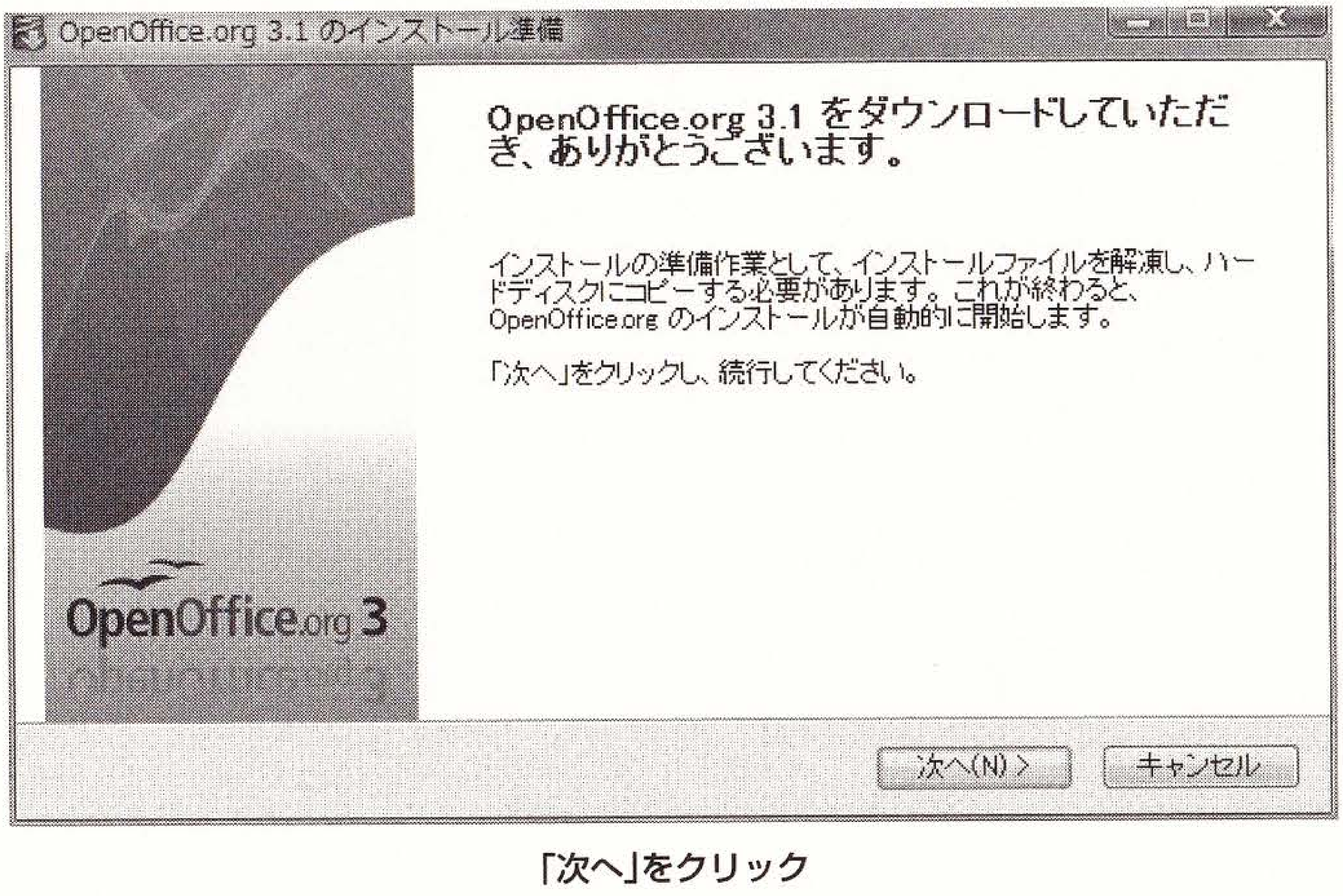
OpenOffice.org日本語プロジェクト
http://ja.openoffice.org/

[2]ダウンロードのページから、利用しているOSに対応したインストーラを選び、ダウンロードします。



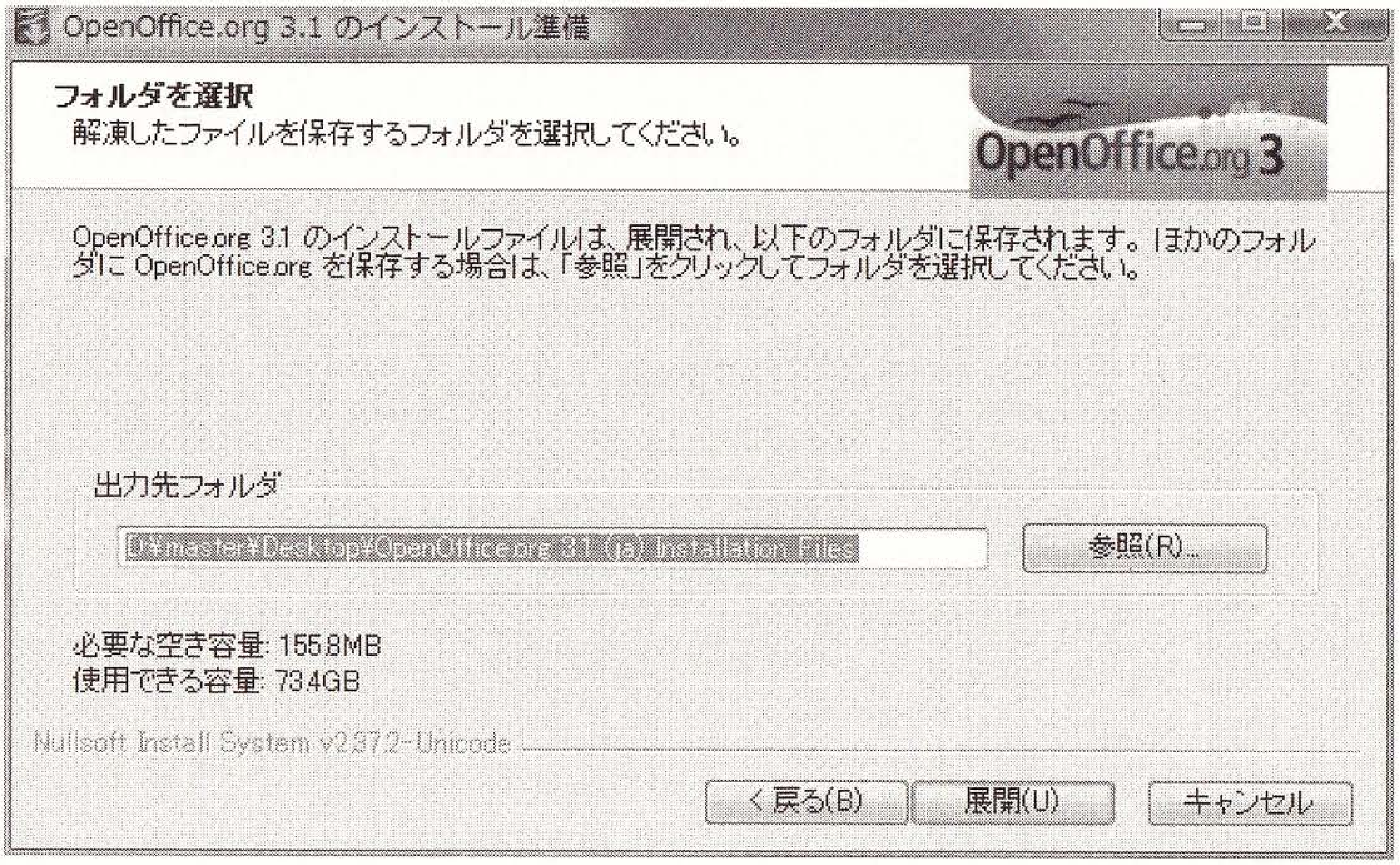
対応OSのインストーラを選んで、ダウンロード

[3]インストーラのダウンロードが終わったら、インストーラをダブル・クリックしてインストールの準備を開始します。



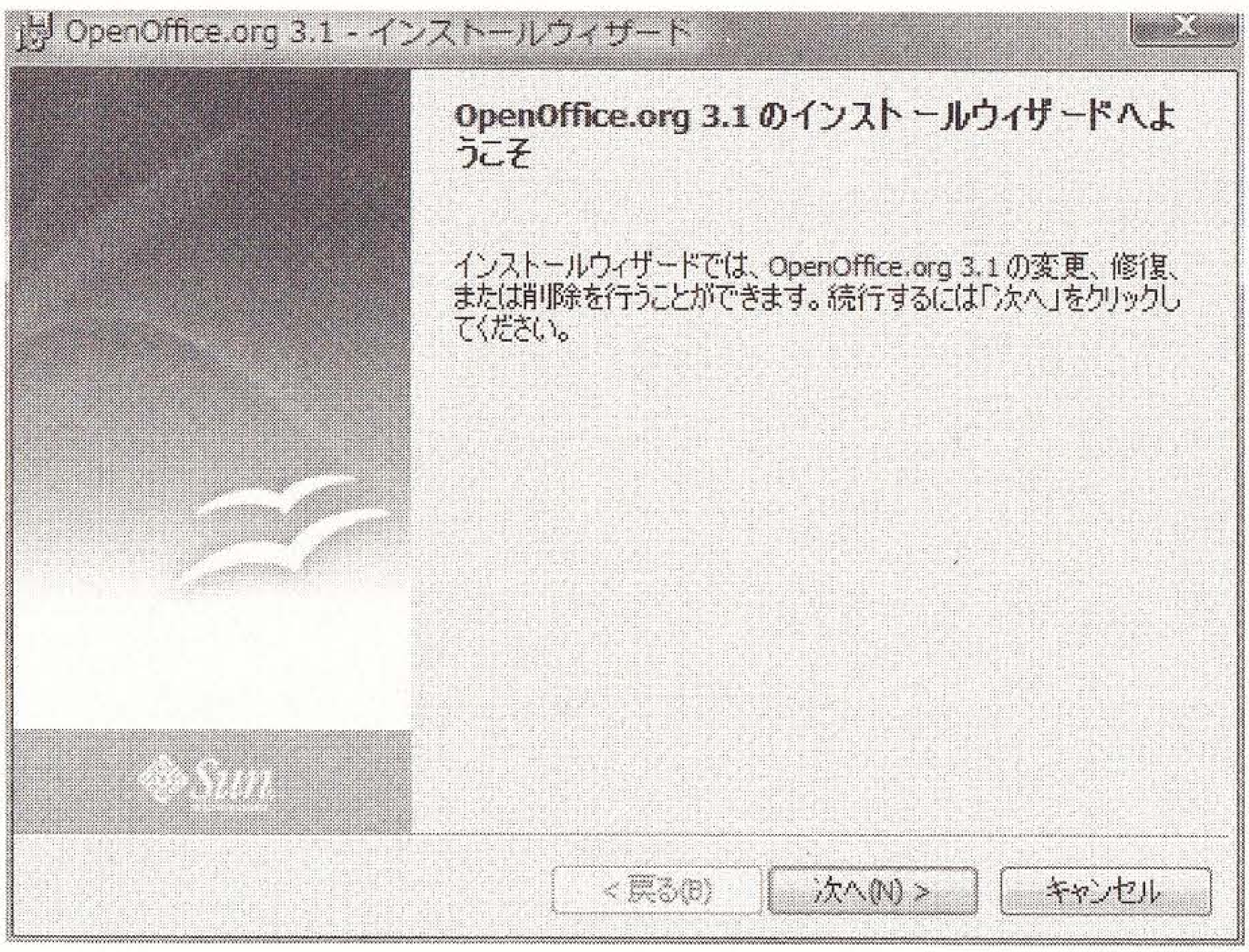
「次へ」をクリック

[4]ファイルの解凍先フォルダを指定する画面が表示されますが、もし解凍先フォルダを変更する場合は、[参照]をクリックして、フォルダを指定してください。



特に変更の必要がなければ、そのまま「展開」をクリック

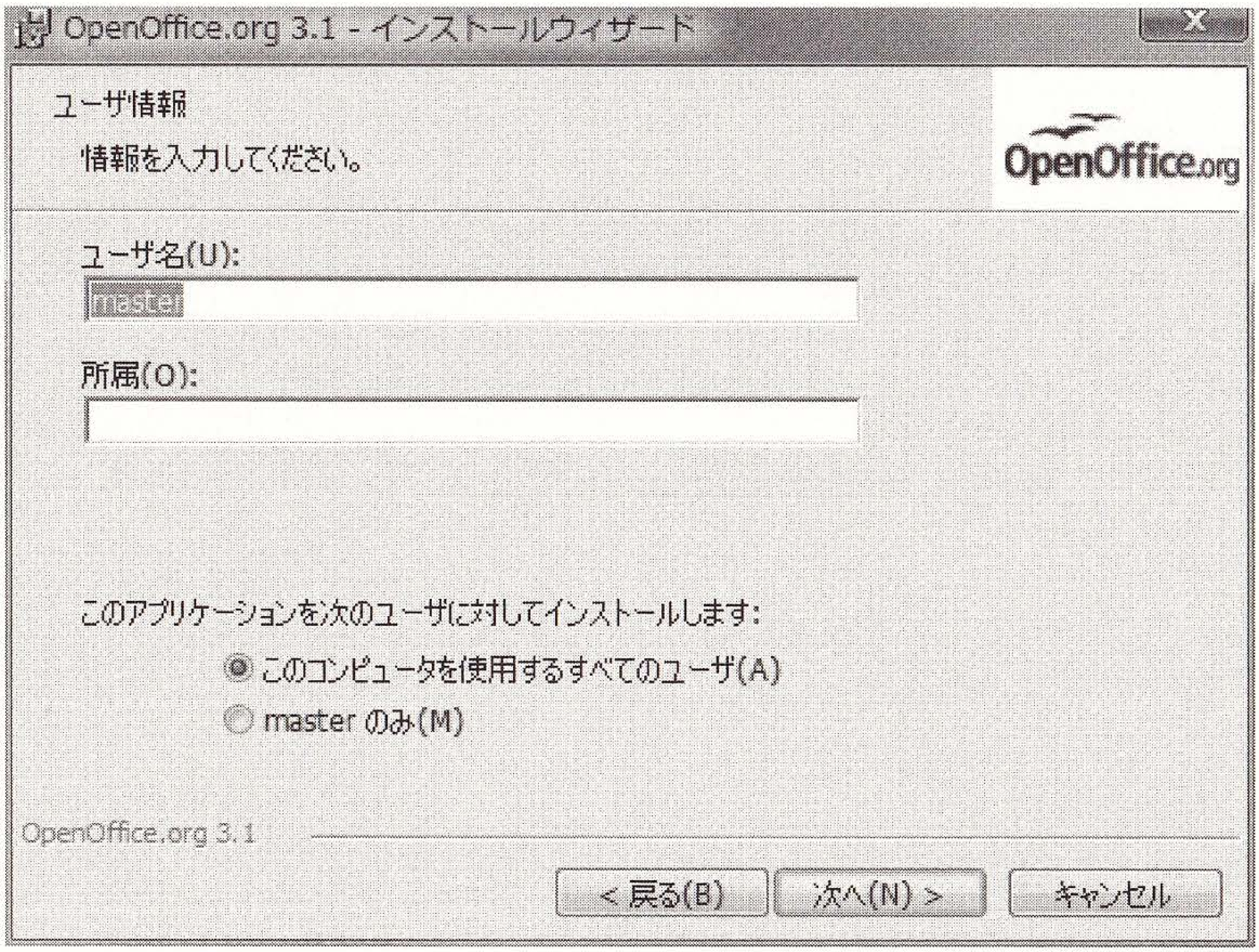
[5]インストール・ウィザードが開始するので、指示に従って作業を進めます。



インストール・ウィザードを開始する

[6] インストールを進めていくと、「ユーザー情報」を登録する画面が表示されます。

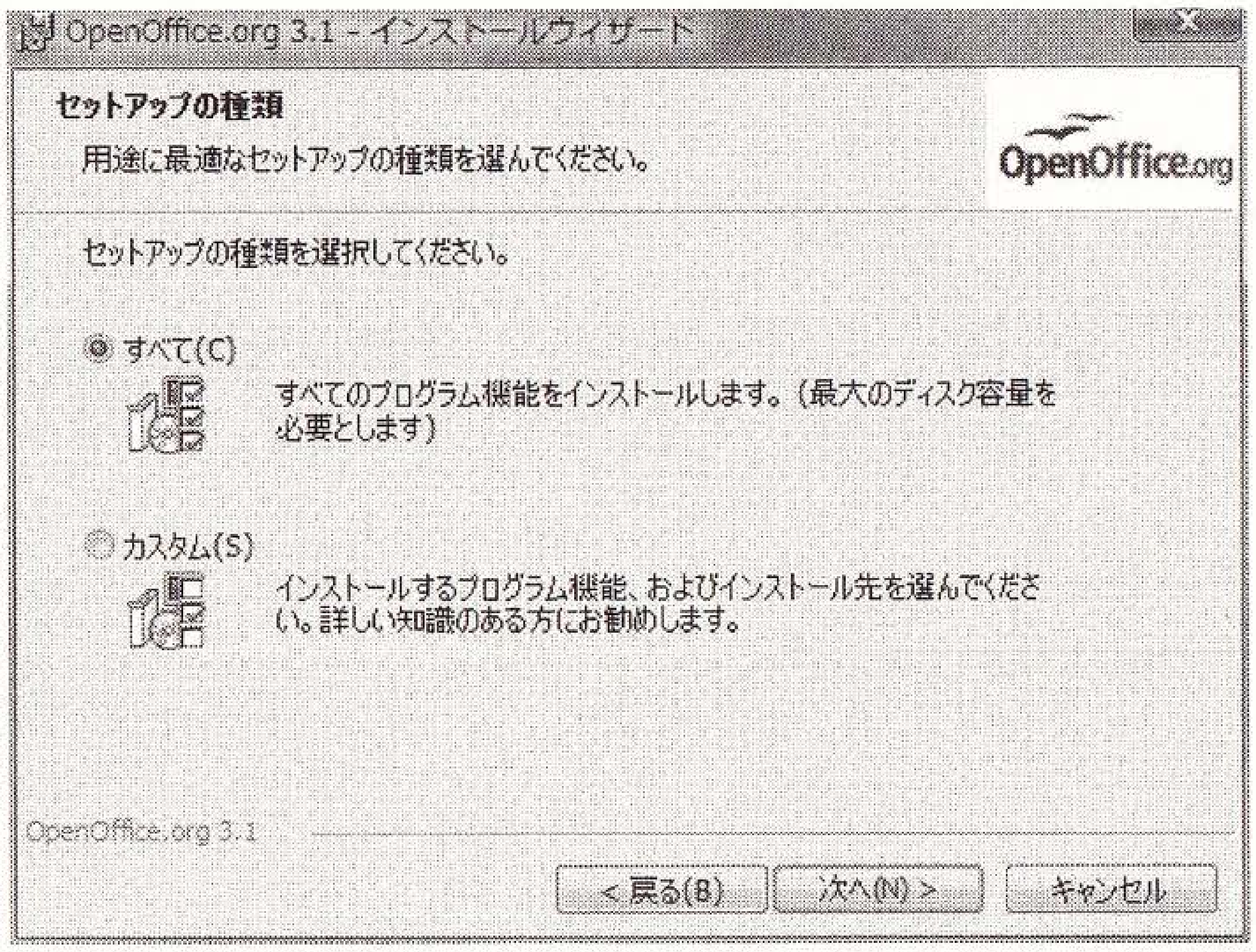
ここでは、「ユーザー名」と「所属」を入力しますが、「所属」は未入力でもかまいません。



「ユーザー情報」を任意で登録する

[7]続いて表示される「セットアップの種類」画面では、「すべて」にチェックを入れて、「次へ」をクリックします。

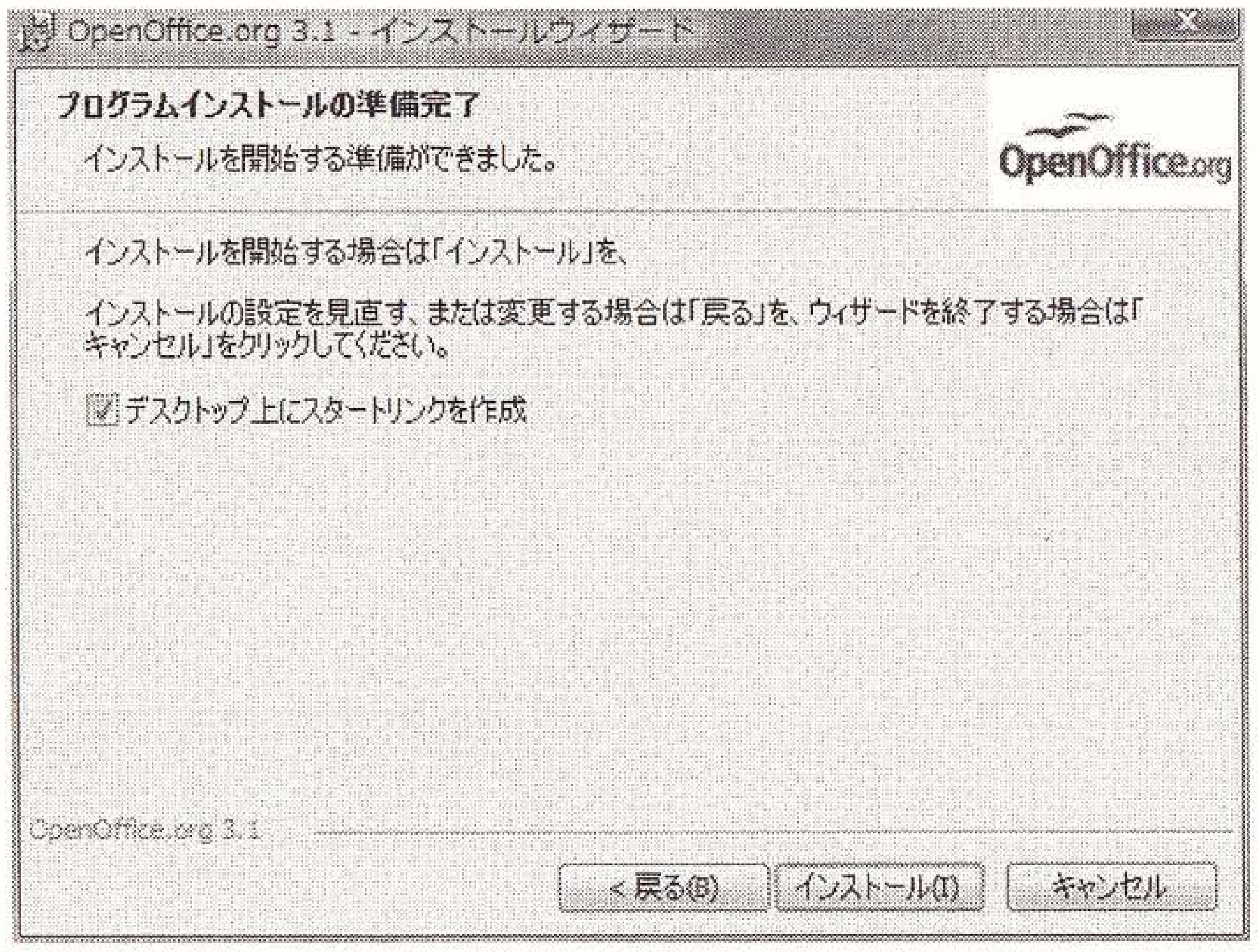
このとき、「カスタム」にチェックを入れると、インストールするプログラムやインストール先を指定できます。



「セットアップの種類」を選択

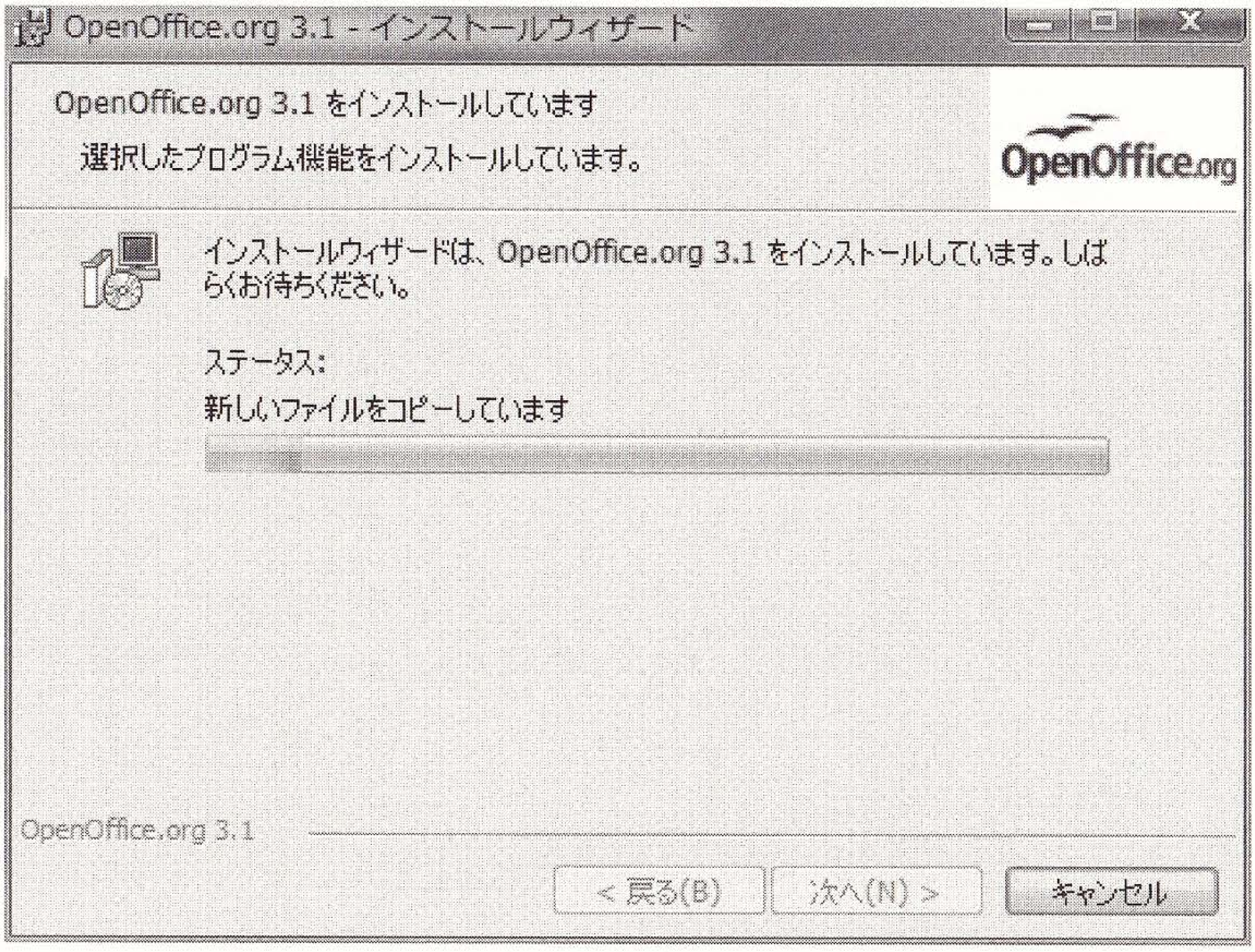
[8]「インストールの準備完了」画面が表示されます。

「デスクトップ上にスタートリンクを作成」にチェックを入れると、デスクトップにショートカット・アイコンが作られるので、必要な人はチェックを入れておきましょう。



ショートカット・アイコンの有無を選択

[9]「インストール」をクリックすると、インストールが開始します。



インストールが開始する

■JRE(Java Runtime環境)

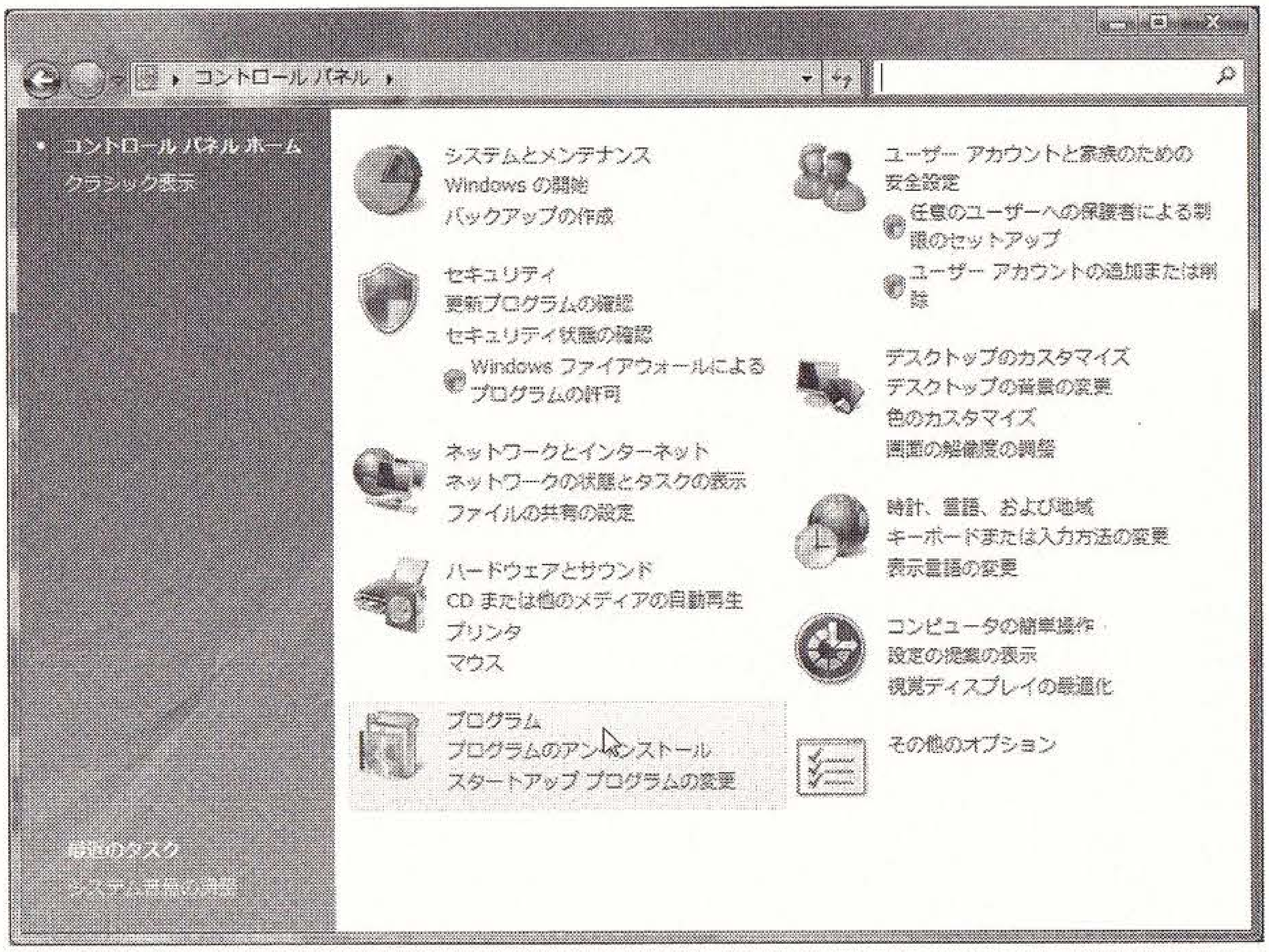
「OOo」の機能の中で、「Writer」や「Calc」の一部の機能や、「Base」の実行を行なうには、「JRE」(Java Runtime環境)が必要になります。

使っているパソコンにJavaがインストールされていない場合は、「JRE付き」をダウンロードしてください。

使っているパソコンに「JRE」がインストールされているかが分からない場合は、次の手順で確認することができます。

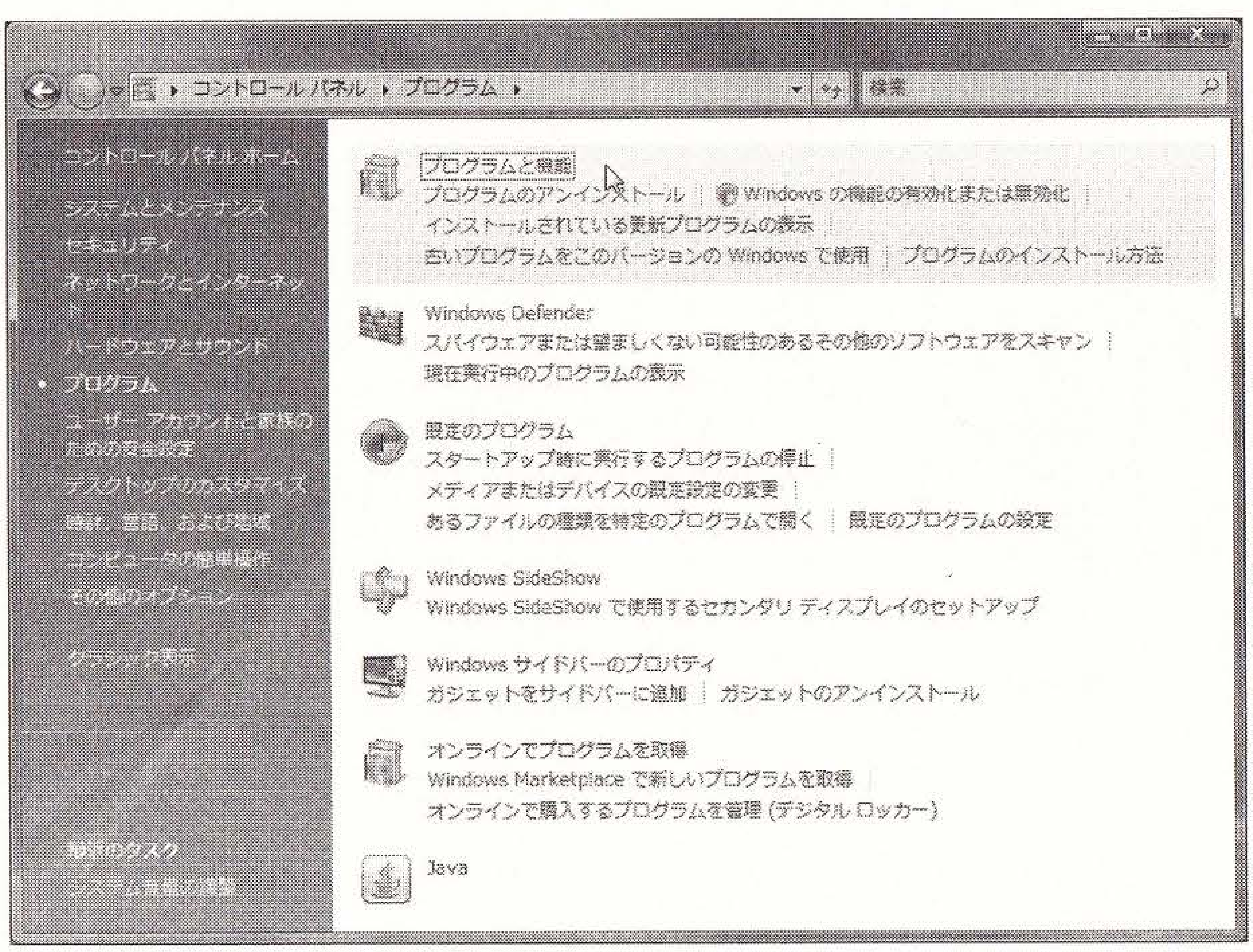
【手順】

[1]「コントロールパネル」を開きます。



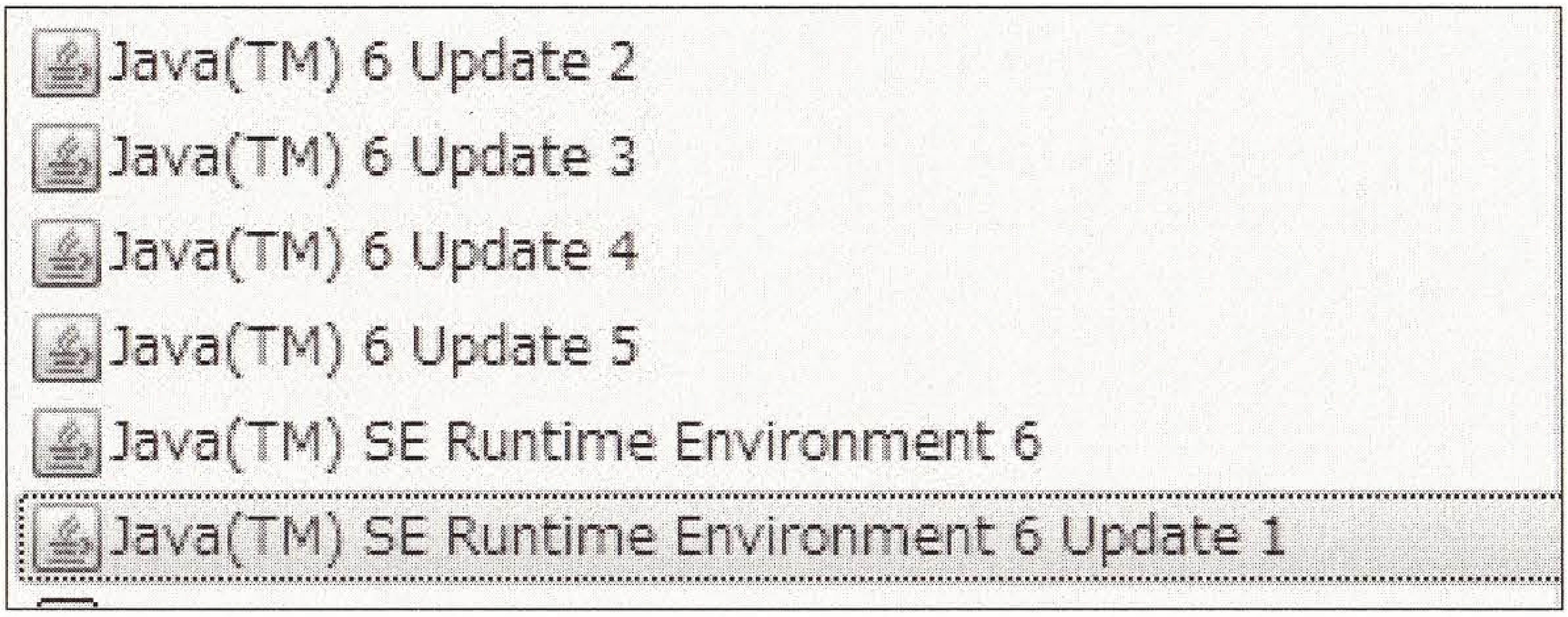
コントロールパネル画面を開く

[2]「プログラム」→「プログラムと機能」を選択します。



「プログラム」の項目から「プログラムと機能」を選択

[3]一覧に「Java(TM) SE Runtime Environment」が表示されているかを確認します。



「Java(TM) SE Runtime Environment」という項目があるか確認

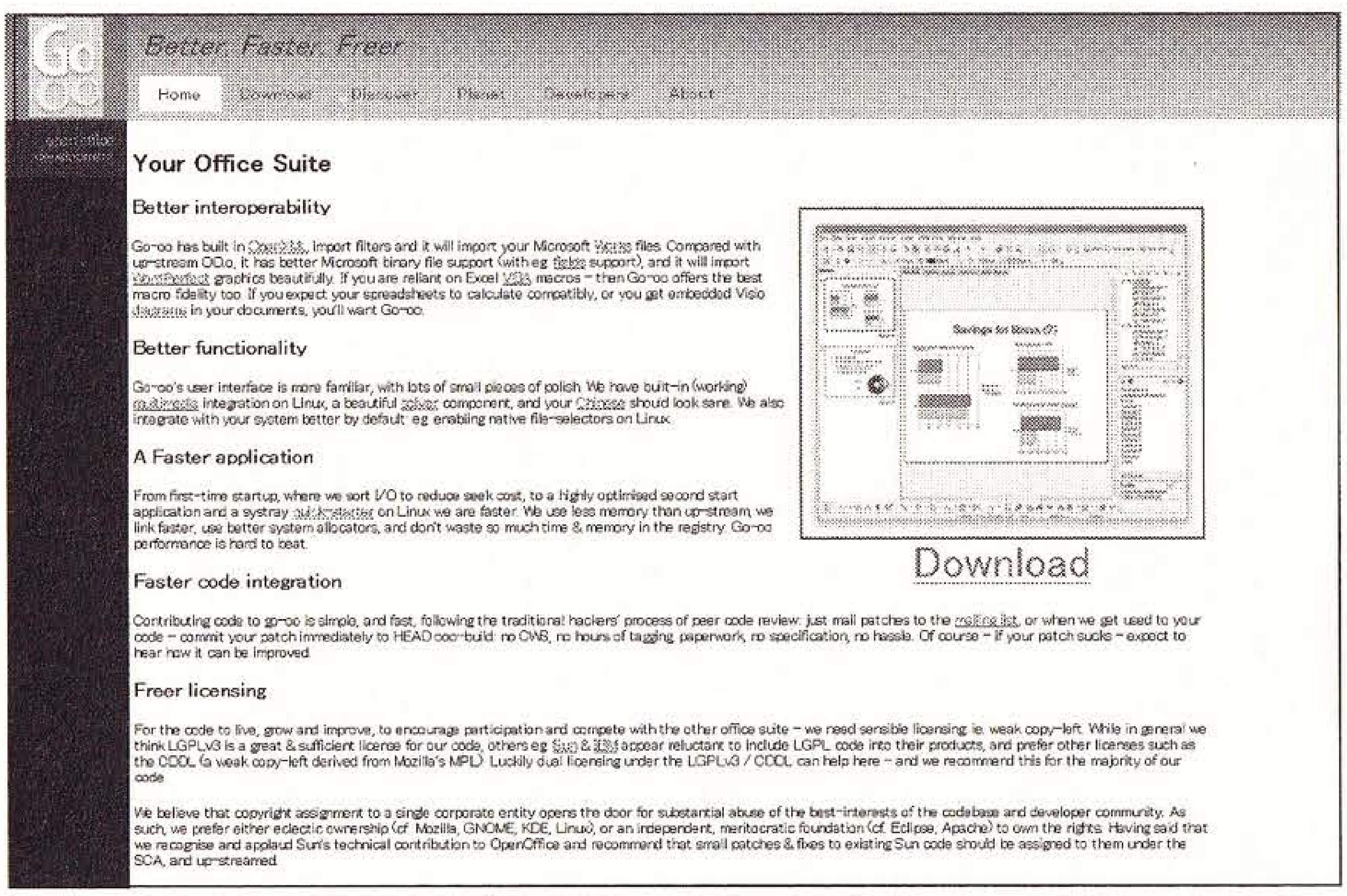
表示されていれば、すでにインストールがすんでいる状態なので、「JREなし」のインストーラでインストールしても、「OOo」の機能を余すところなく使えます。

付録2 「OpenOffice.org」の派生ソフト「Go-OO」

「OOo」には、「StarSuite」や「Lotus Symphony」など、数々の派生ソフトがあります。

その中で、「OOo」上で、「VBAマクロ」をサポートするために開発が続けられているのが、「Go-OO」(ジーオー・オーオー)と呼ばれるソフトです。

対応OSは、「Windows」「Mac」「Linux」です。



Go-OOのWebサイト
http://go-oo.org/

また、「VBAマクロ」のサポートのほか、高速な起動や「MS Works」のサポート、「WordPerfect」のグラフィックスのインポートなど、さまざまな機能が追加されています。

■「Go-OO」のインストールと日本語化

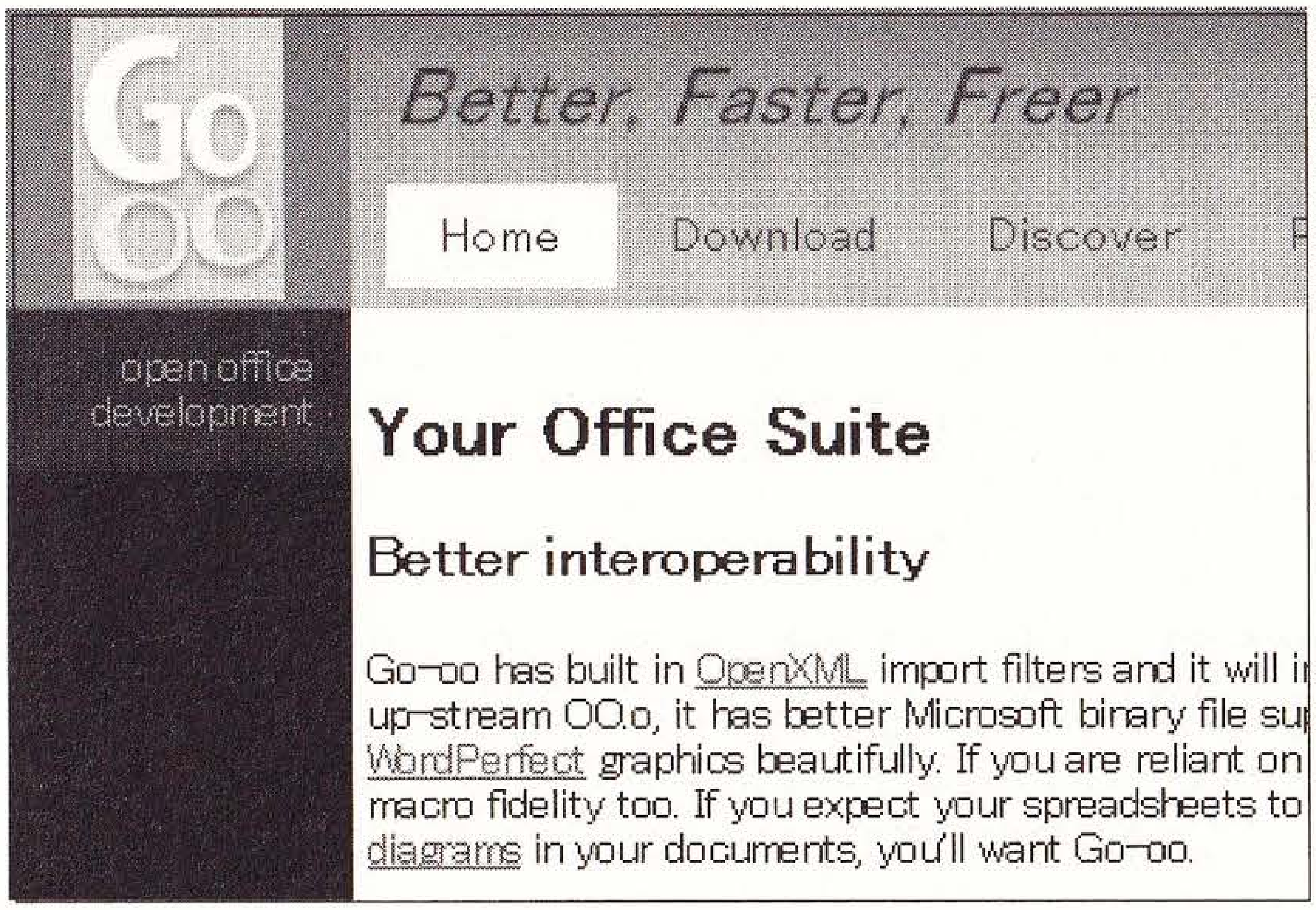
「Go-OO」は、プログラム上では「OOo」として扱われるので、公式版「OOo」との共有はできません。

また、日本語化のためのランゲージパックを別途インストールする必要があります。

インストールと日本語化の手順は以下のとおりです(Windows版の場合を例にしています)。

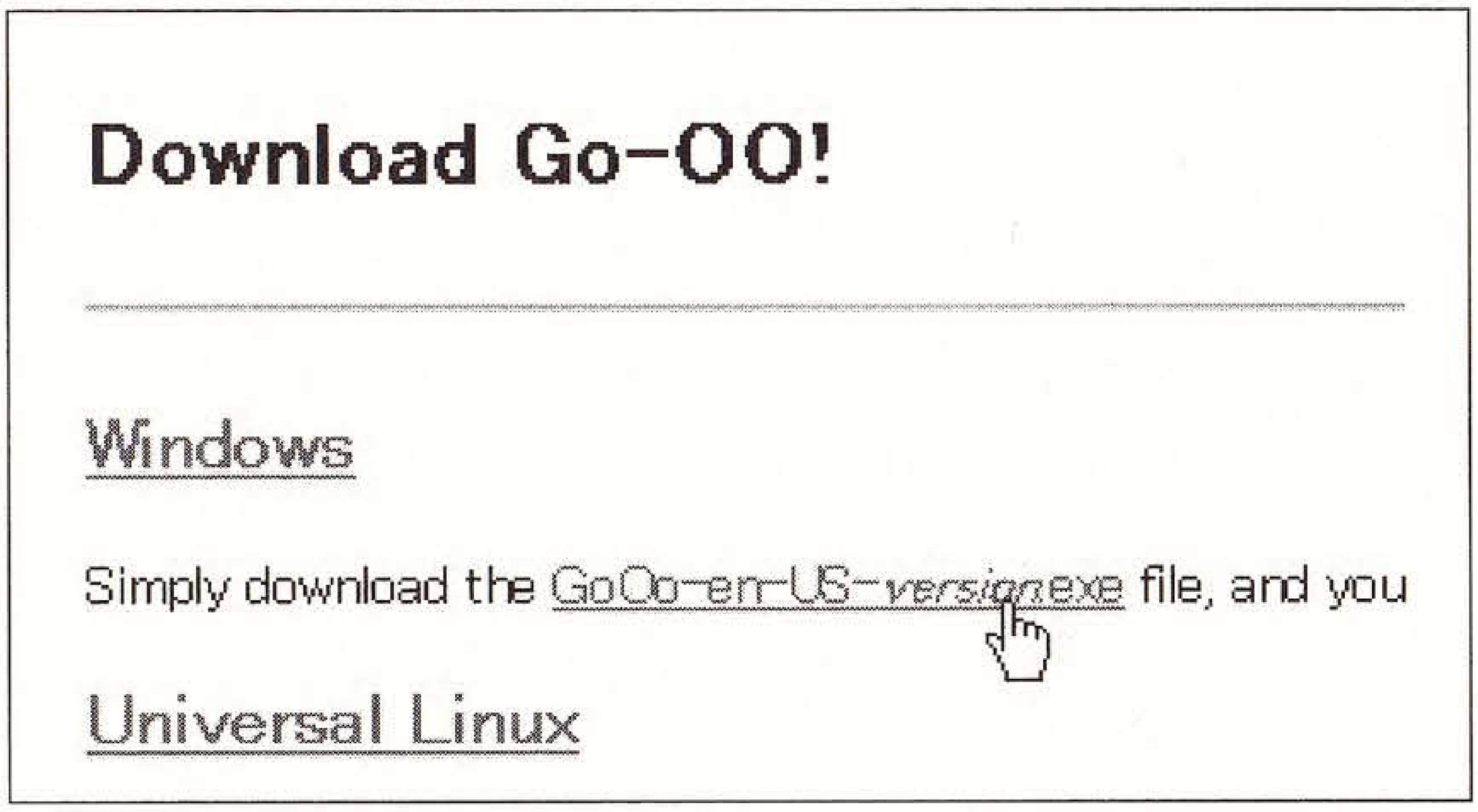
【手順】

[1]「Go-OO」のWebサイトから、トップページ上部にある「Download」の項目をクリックします。



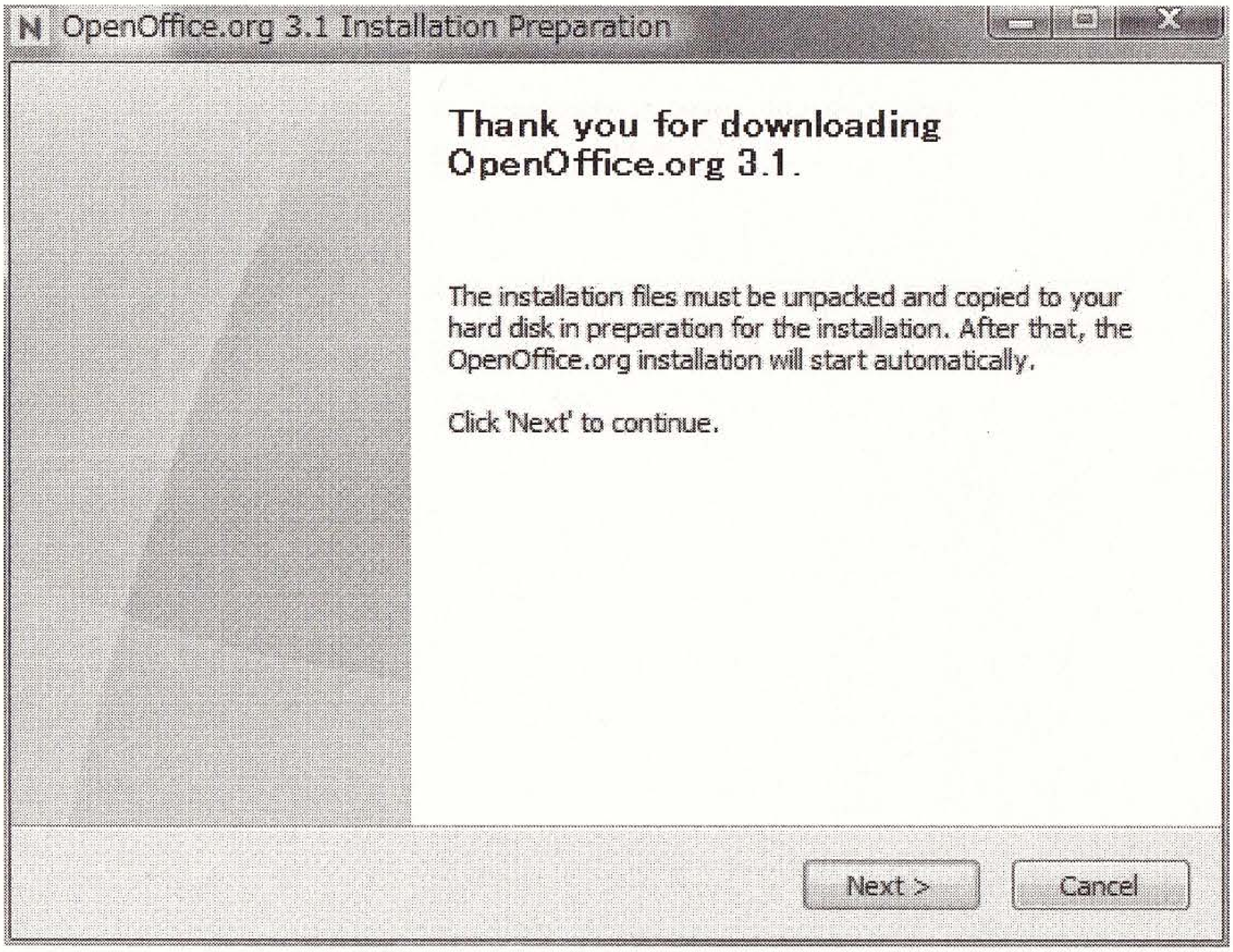
「Download」をクリック

[2]利用しているOSに対応したインストーラをダウンロードします。今回は、Windows版の「Go-OO US版」をダウンロードします。
(念のため、インストール前に、ウイルスチェックは必ず行なっておきましょう)。



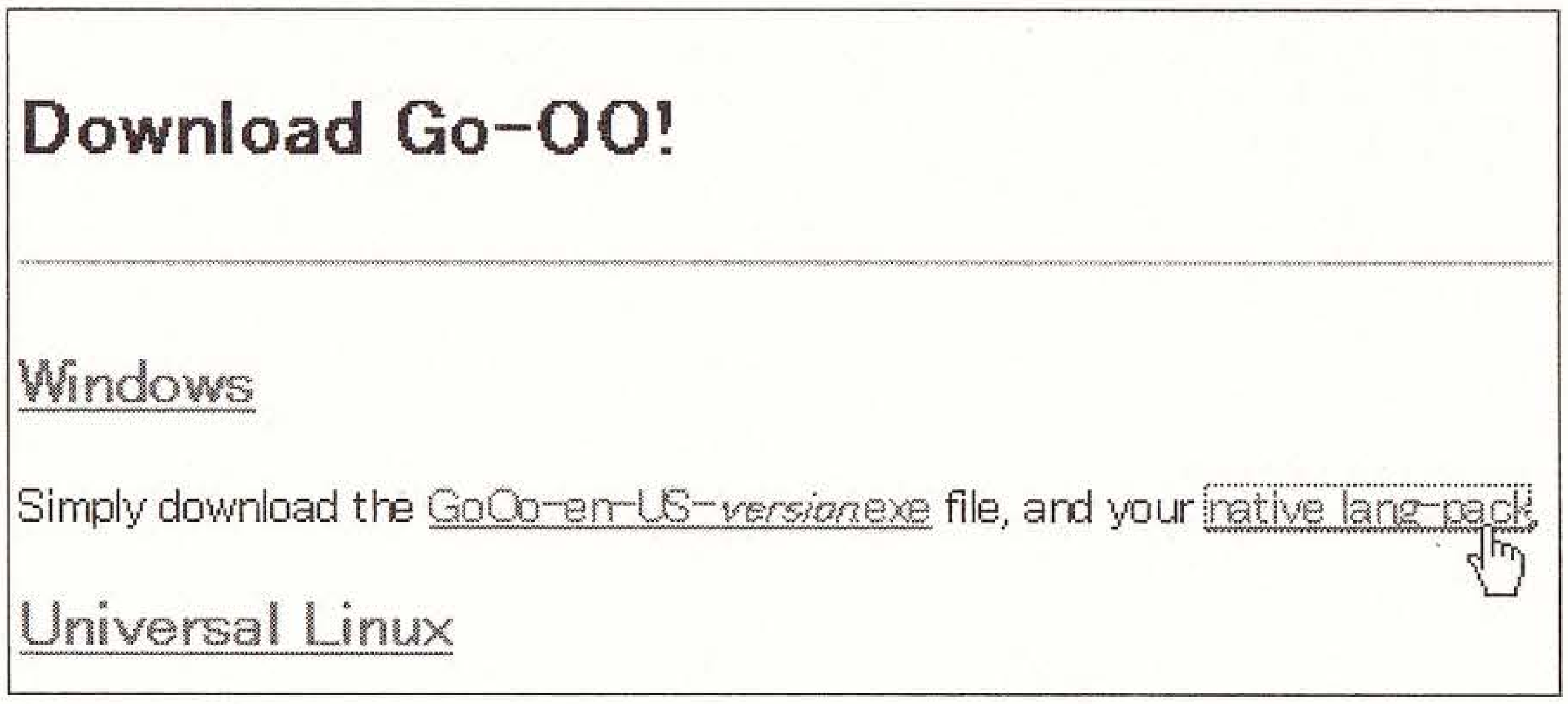
「GoOo-en-US-version.exe」をクリックして、インストーラをダウンロード

[3]ダウンロードしたファイルをダブル・クリックして、インストールを開始します。インストール・ウィザードは英語で表記されますが、手順は公式版「OOo」と同じです。



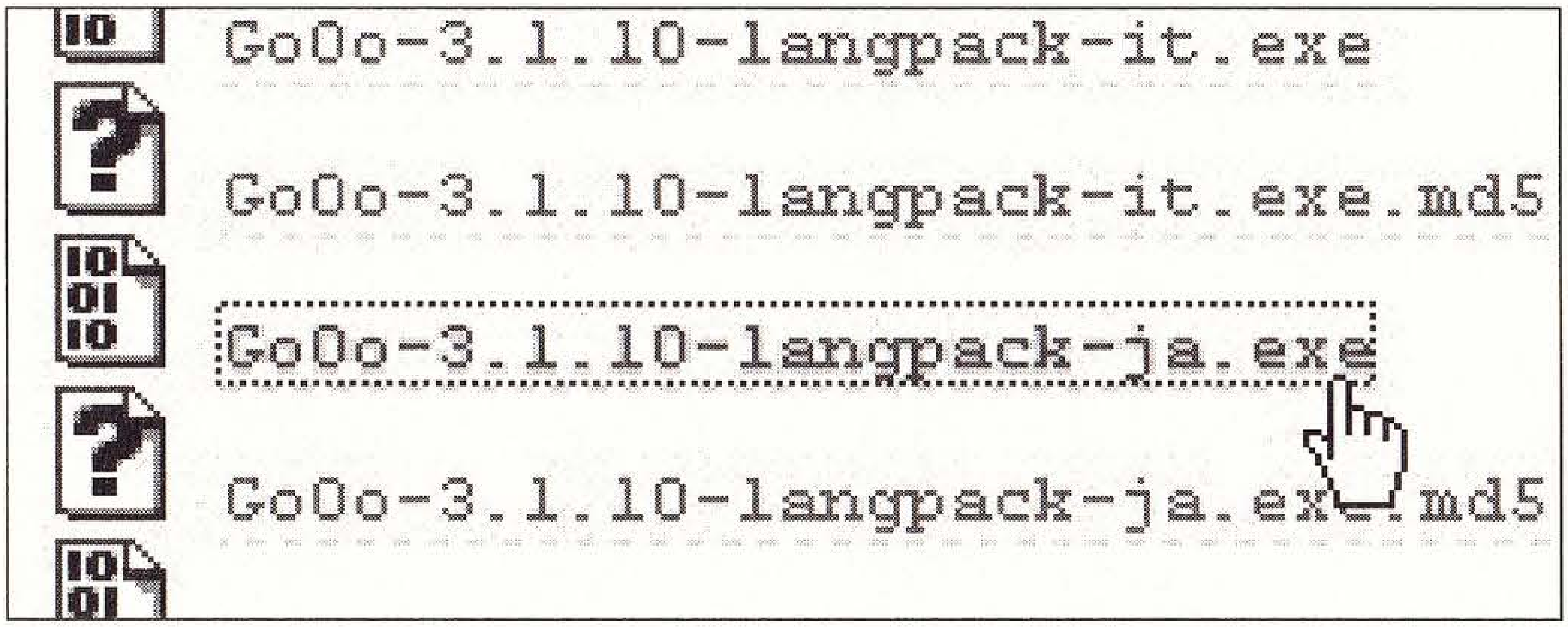
インストール手順は公式版と同じ

[4]インストールが終了したら、日本語化の作業に移ります。
まず、ダウンロード・ページから「native lang-pack」をクリックします。



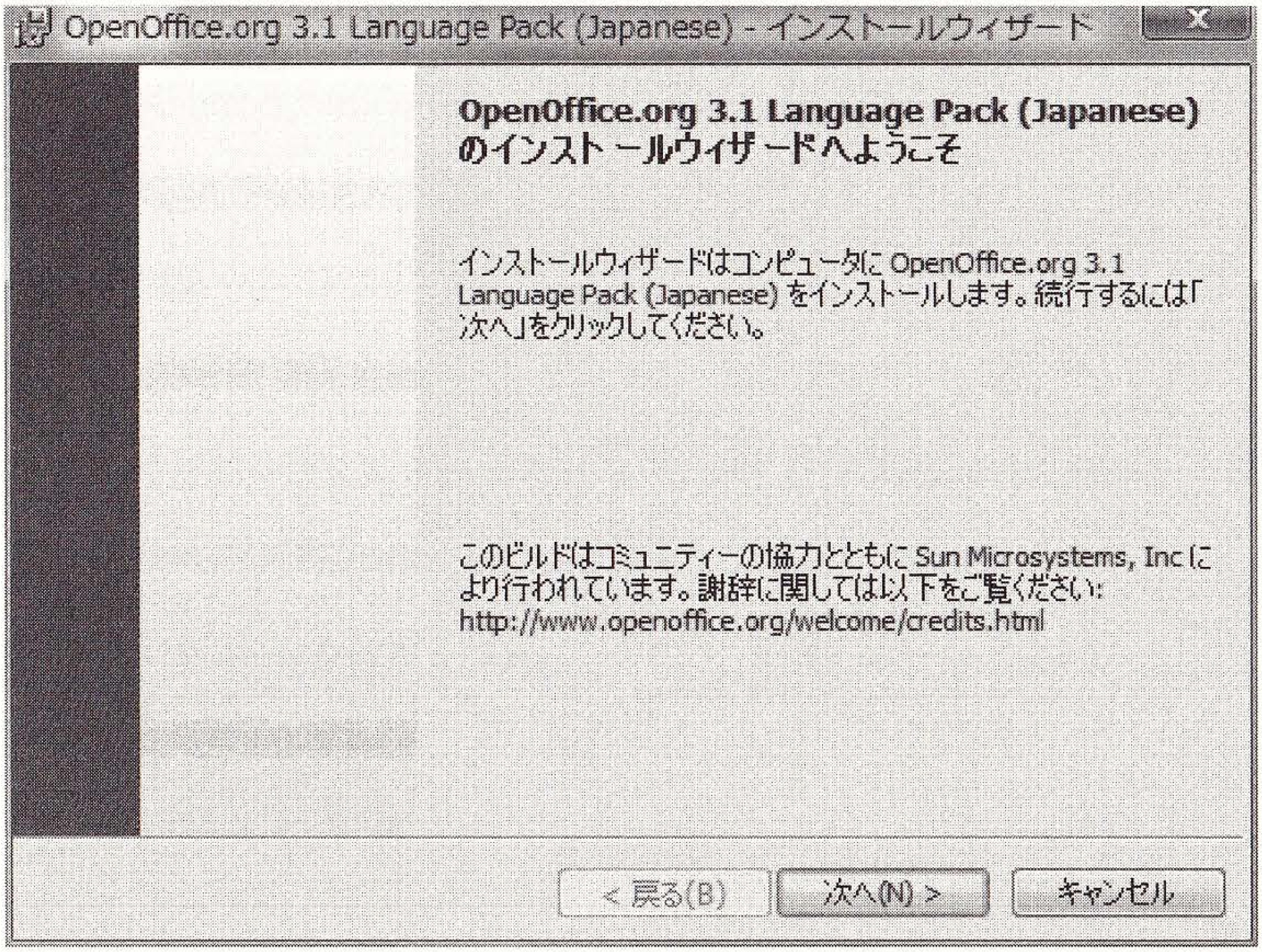
「native lang-pack」をクリック

[5]ランゲージパックのディレクトリ・ページに移動するので、「GoOo-3.1.10-langpack-ja.exe」をクリックし、ファイルをダウンロードします。



日本語のランゲージパックをダウンロード

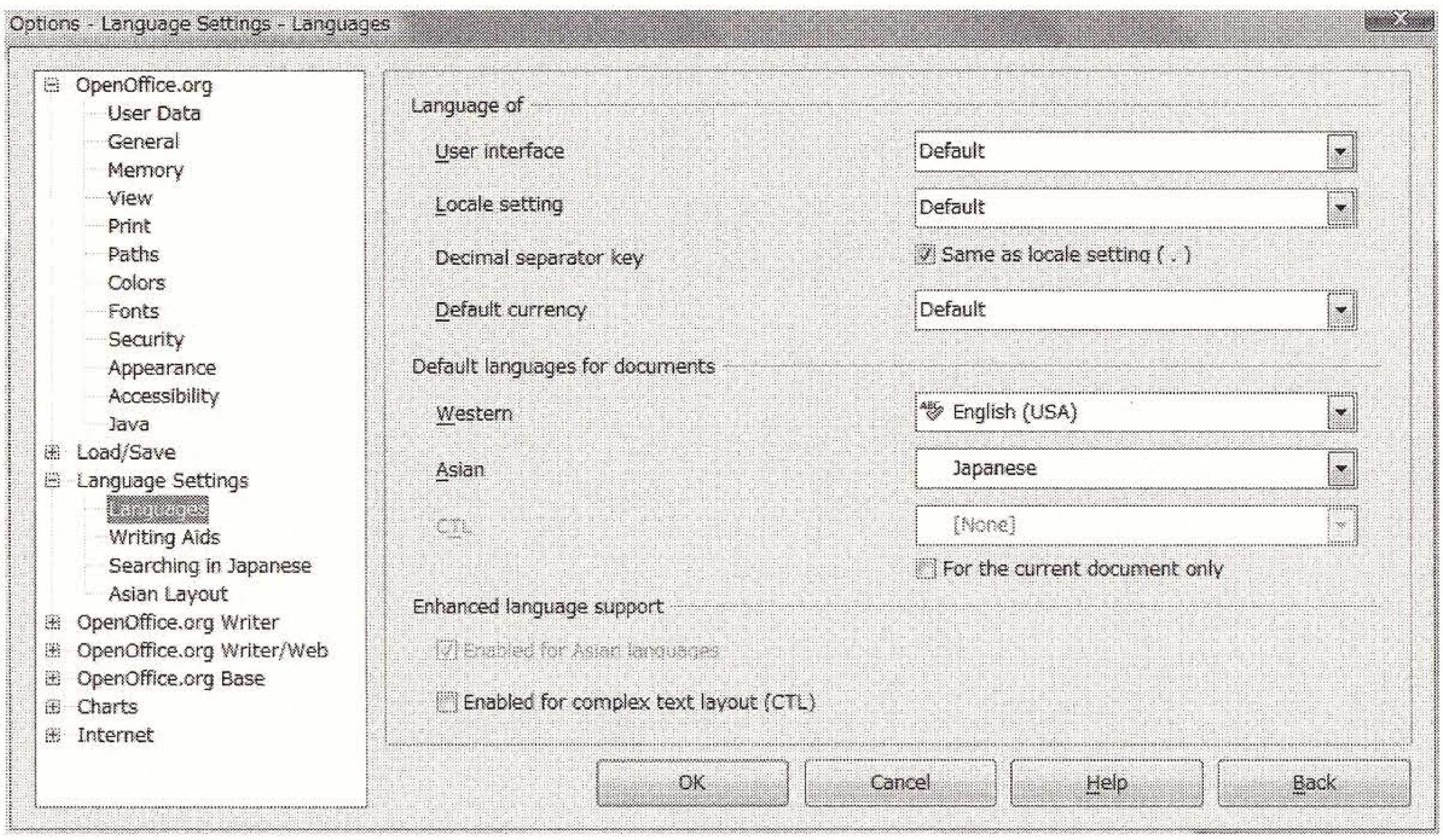
[6]ダウンロードしたファイルをダブル・クリックして、画面の指示に従ってインストールします。



日本語ランゲージパックをインストール

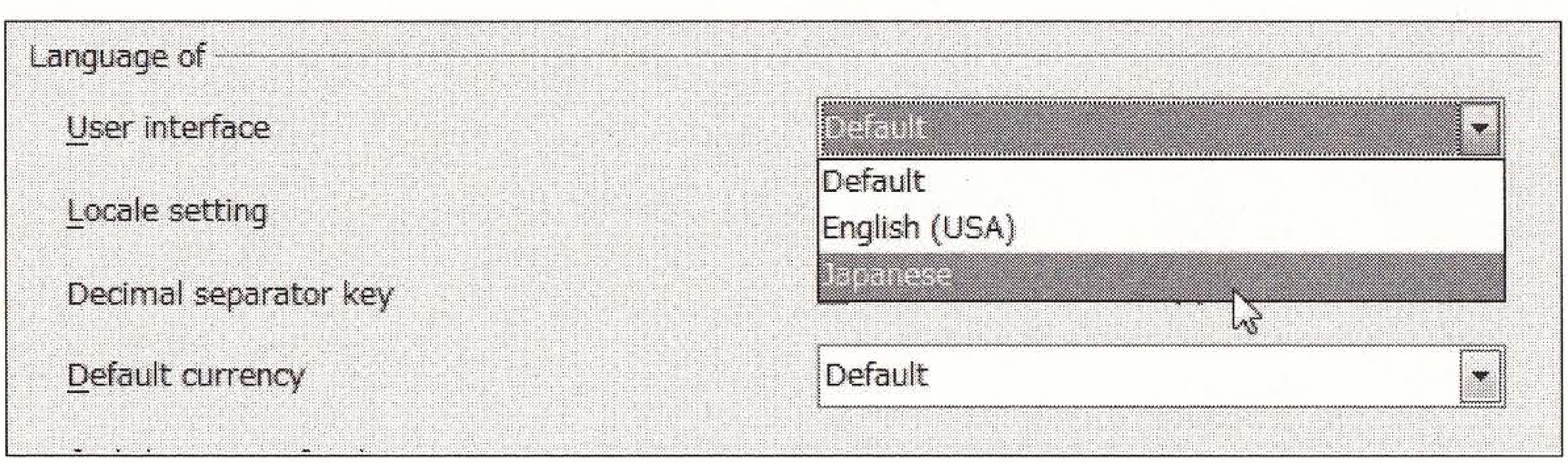
[7]インストールが終了したら、「Go-OO」を起動します。

適当なアプリケーションを起動して、「Tool」→「Option」をクリックしすると、オプション設定画面が表示されるので、「Language Settings」→「Language」を選択します。



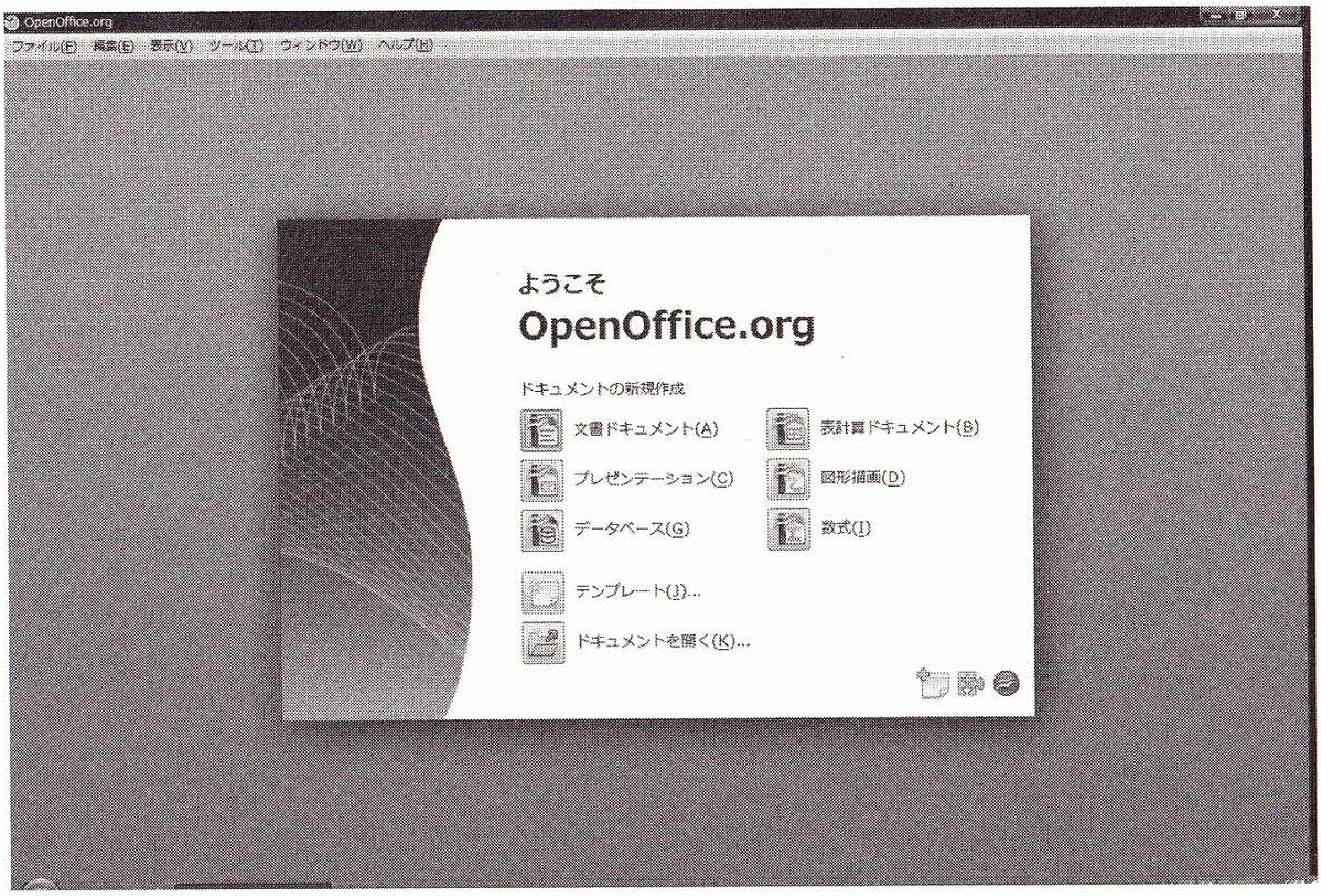
「Language Settings」→「Language」を選択

[8]「Language of」の項目にある「User interface」「Locale Setting」を、「Japanese」に設定します。



言語の基本設定を「日本語」に設定する

[9] あとは、「Go-OO」を再起動すれば、日本語化は終了です。



日本語化が完了する

付録3　困ったときのために

「OOo」でマクロを組んでいると、構文や具体的な処理などの疑問がいろいろ出てくると思います。

そのようなときは、以下の資料で疑問を解決できないか、調べてみてください。サーチ・エンジンの検索機能とあわせて調べてみれば、たいていのことには「ヒント」があり、さらには「回答」も得られると思います。

●「OOo」のヘルプ

「ヘルプ」メニューにある「OpenOffice.orgヘルプ」には、「マクロとプログラミング」の項目があって、「基本的な構文」や「組み込みのサブルーチンや関数」が紹介されています。

あまり詳しく書いてあるわけではありませんが、概要を掴むのには役に立つでしょう。

具体的な処理を浮かんできた段階で、“こうした処理はどういった構文で書けばいいのだろう”といった疑問がわいてきたときは、まずはヘルプを調べてみてください。

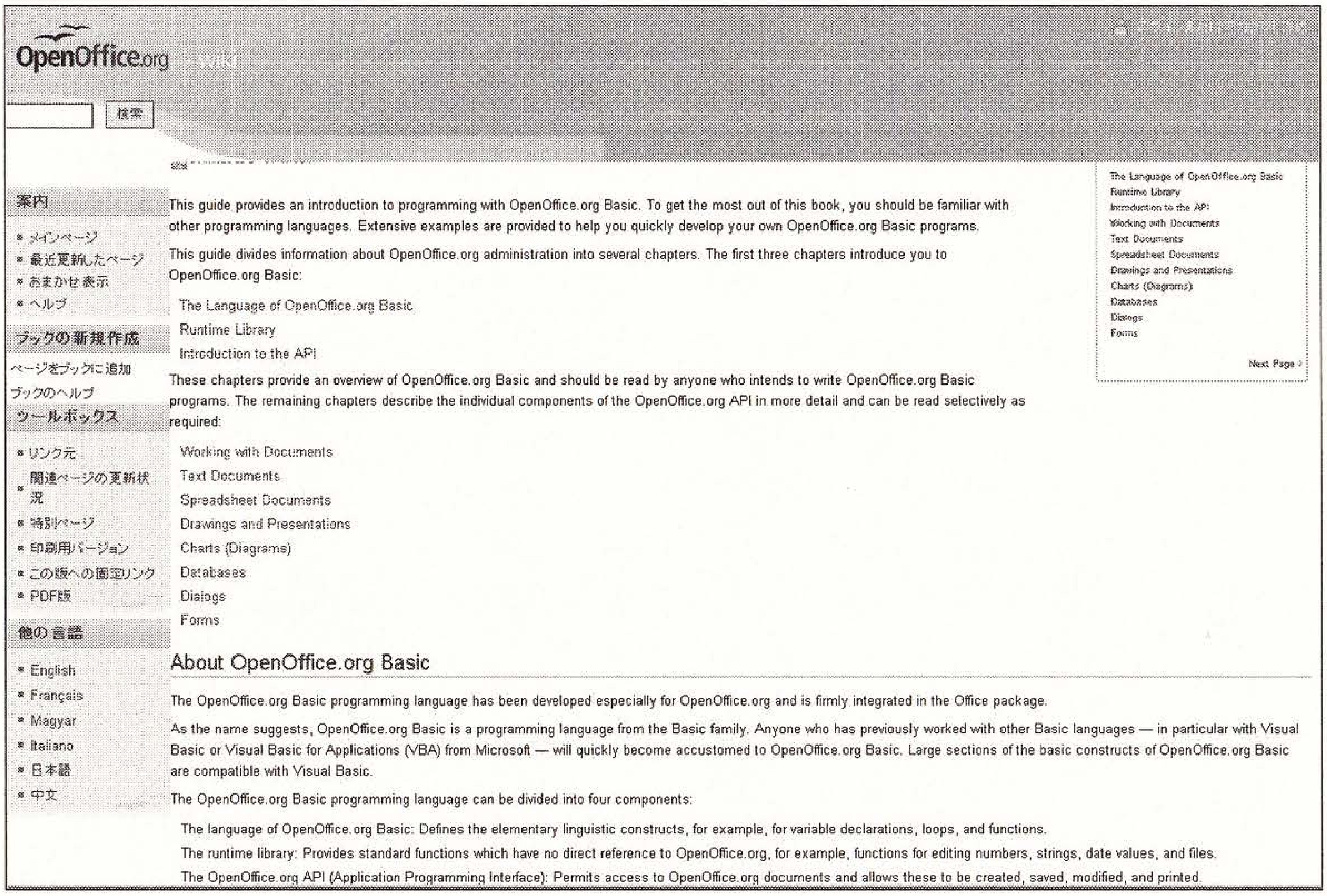
また、関数は正常に処理を行なえないとエラーの値を返しますが、その値(エラー・コード)についても解説されているので、おかしな動作をする関数があったら、その返り値を調べてみると解決のヒントになるかもしれません。

●OpenOffice.org BASIC Programming Guide

http://wiki.services.openoffice.org/wiki/Documentation/BASIC_Guide

「Basicマクロ」のプログラミング・ガイドです。英語ですが、かなり充実した内容なので「Basicマクロ」にある程度慣れてきたら、一通り目を通してみてください。

「Calc」に関する事柄は、「Spreadsheet Documents」にまとめられています。



OpenOffice.org BASIC Programming Guide

●APIリファレンス

http://api.openoffice.org/docs/common/ref/com/sun/star/module-ix.html

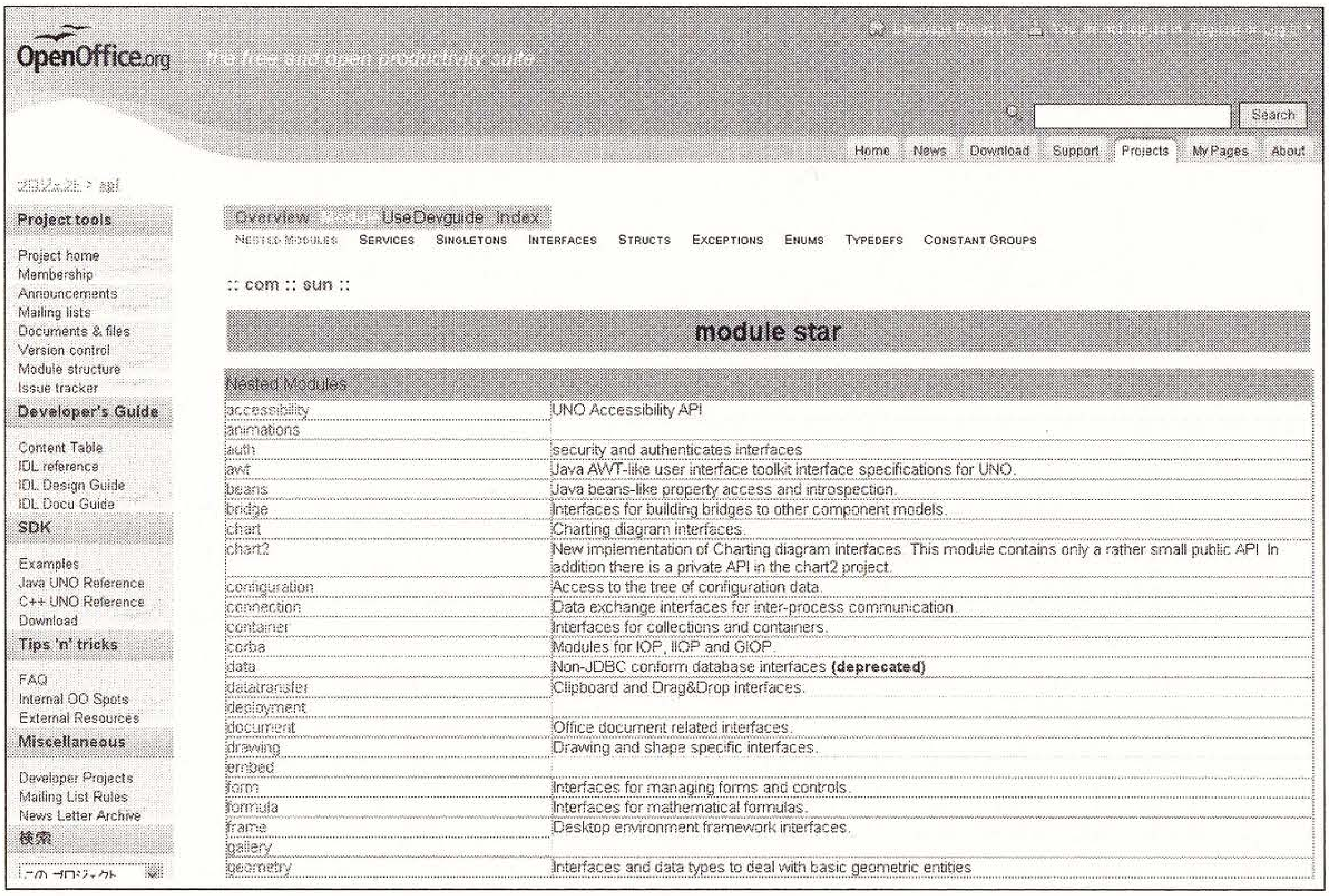
マクロなどで使える、「OOo」のAPIリファレンスです。

英語ですが、各APIの引数や機能の概略がまとめられているので、本格的に「OOo」の機能を活かしたマクロを組むには必須の資料でしょう。概略が簡潔にまとめられ、比較的読みやすいと思います。

また、「OOo」のAPIは機能ごとに階層化されているので、欲しい機能があるときは、上の階層から関連のありそうな名前のリンクをたどってみてください。

このほか、サーチ・エンジンでネット上に公開されているサンプルを探すのも有効です。

関連のない結果が列挙されて絞り込めないときは、「OOo」の「Basicマクロ」で使いそうな単語(「OpenOffice.org Basicマクロ」の各種構文や「ThisComponent」など)をキーワードに含めてみましょう。



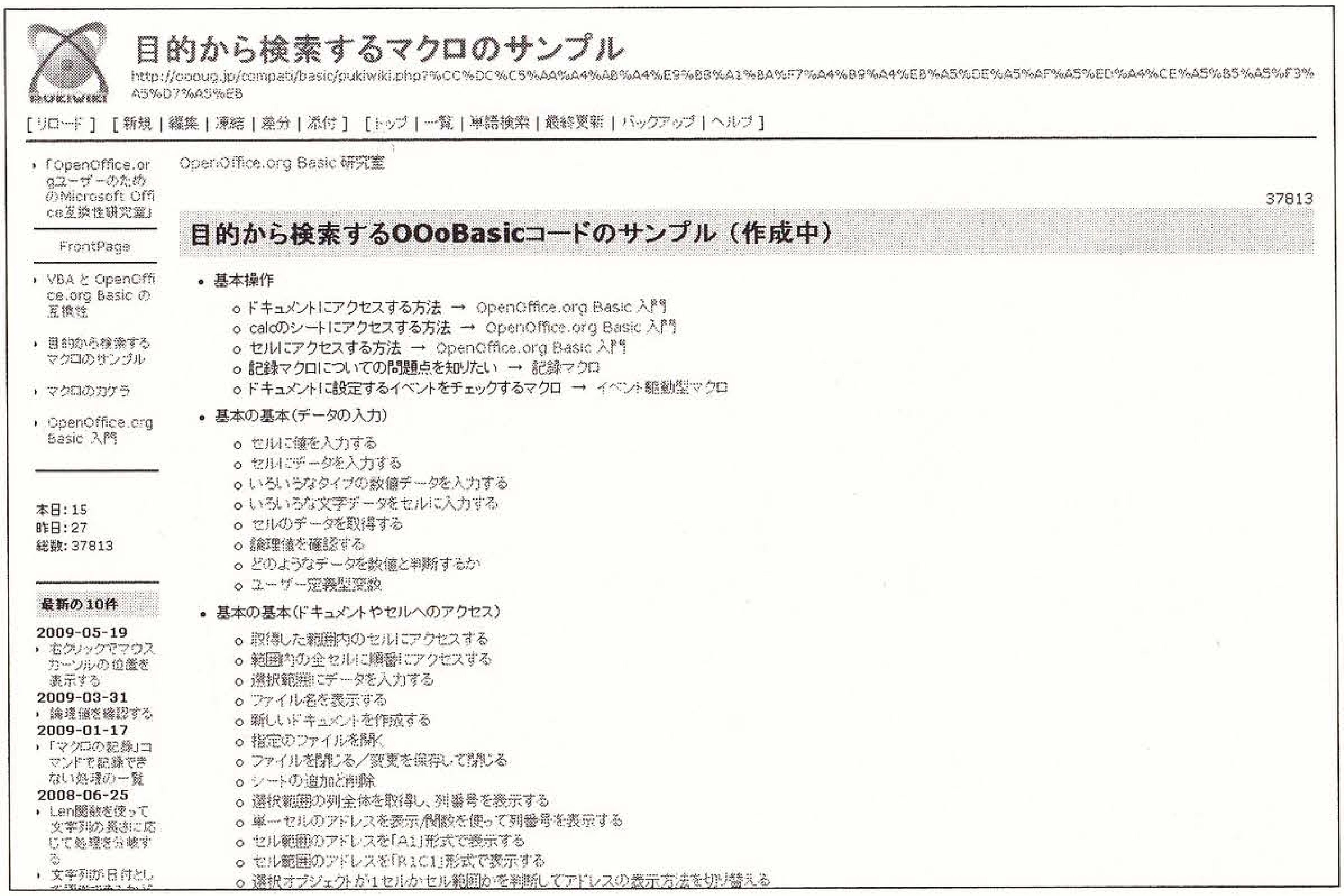
APIリファレンス

●OpenOffice.org Basic 研究室

http://ooug.jp/compati/basic/pukiwiki.php?OpenOffice.org%20Basic%20%B8%A6%B5%E6%BC%BC

「OpenOffice.org Basicマクロ」のサンプルを公開しているページです。

サンプルの数はそこまで多くはありませんが、プログラミングの書き方に迷ったときなどに、目的の用途と似たサンプルがあれば、参考になるかもしれません。



OpenOffice.org Basic 研究室

MEMO

付録4 「OpenOffice.org Basic」予約語一覧(本書関係分)

本書サンプル・プログラム中で使われている予約語の一覧です。

As	ByVal	Close
date	Dim	Do
Double	Else	End
For	Function	If
Input	Integer	Line
Long	Loop	Name
New	Next	not
Object	Open	ReDim
Step	String	Sub
Then	to	Until
Wend	While	

索　引

アルファベット順

50音順

記号・数字

[著者略歴]

宍戸　輝光 (ししど・てるみつ)

1973年生まれ。つくば市在住のプログラマー。
半引きこもり状態で、プログラムを作り散らかしている。
最近は、携帯ゲーム関連の開発が中心。

[著者HP] http://www.programmingmat.jp/

【主な著書】
PC＆ケータイ ゲームプログラミング練習帳
ゲームプログラミング練習帳　(以上、工学社)

質問に関して

本書の内容に関するご質問は、

①返信用の切手を同封した手紙
②往復はがき
③FAX(03)5269-6031
(返信先のFAX番号を明記してください)
④E-mail　editors@kohgakusha.co.jp

のいずれかで、工学社編集部あてにお願いします。
なお、電話によるお問い合わせはご遠慮ください。

サポートページは下記にあります。
【工学社サイト】 http://www.kohgakusha.co.jp/

I/O BOOKS 「Writer」「Calc」「Base」…OOoをBasicで便利に！

OpenOffice マクロプログラミング

平成21年10月10日　初版発行　©2009
平成23年 1月25日　第1版第2刷発行

著　者　宍戸　輝光
編　集　第二I/O編集部
発行人　星　正明
発行所　株式会社 工学社
〒160-0003　東京都新宿区本塩町23番地
第二田中ビル7F
電　話　(03) 5269-2041 (代) [営業]
(03) 5269-6041 (代) [編集]
振替口座 00150-6-22510

※定価はカバーに表示してあります。

[印刷] シナノ印刷(株)

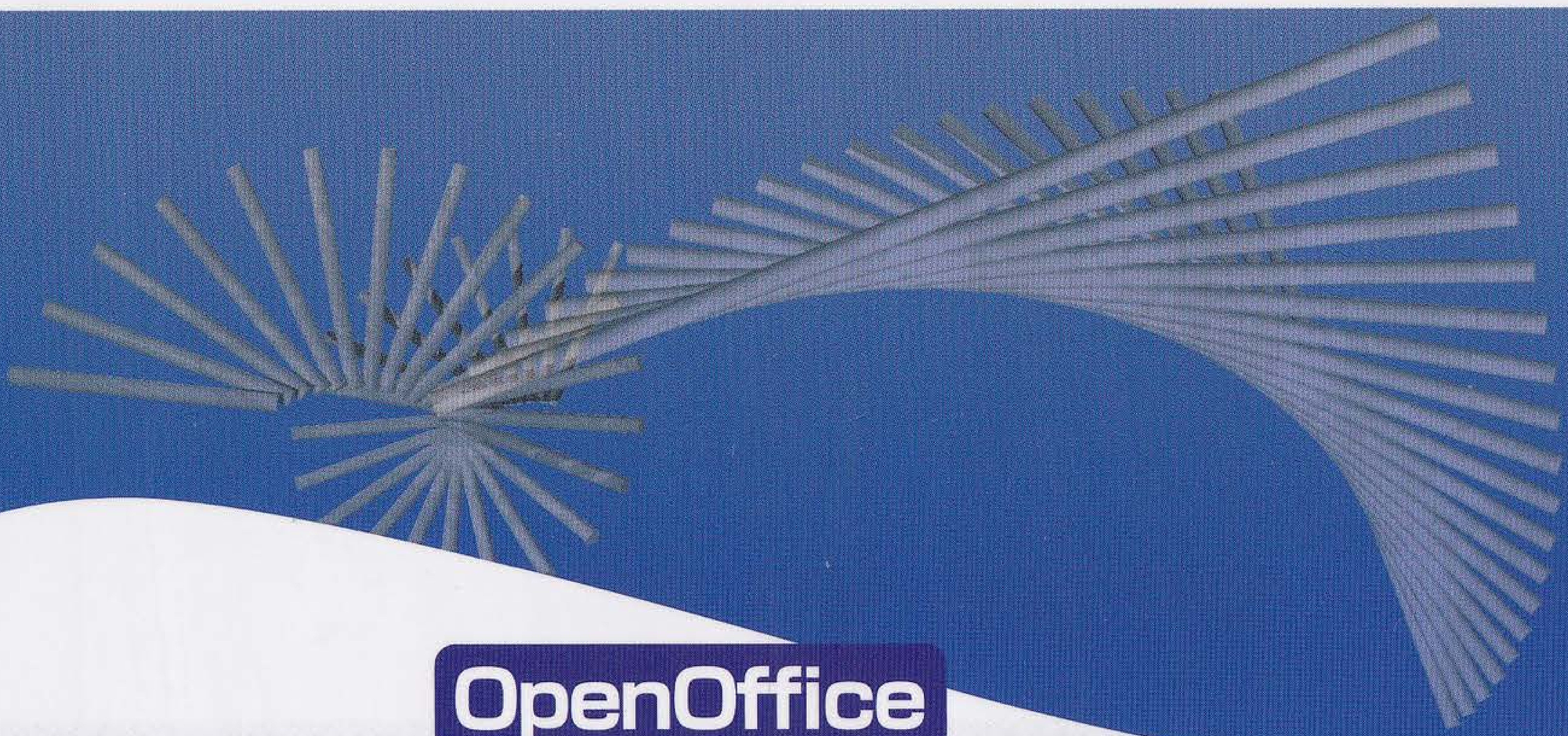
ISBN978-4-7775-1470-0

ISBN978-4-7775-1470-0
C3004 ¥1800E

9784777514700

定価：本体1800円＋税

1923004018002



OpenOffice
マクロプログラミング